Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías Industriales

# Implementación de controladores predictivos en Arduino

Autor: Alfonso García Navarro
Tutor: Daniel Limón Marruedo

**Dep. Ingeniería de Sistemas y Automática**
**Escuela Técnica Superior de Ingeniería**
**Universidad de Sevilla**

Sevilla, 2016

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías Industriales

# Implementación de controladores predictivos en Arduino

Autor:
Alfonso García Navarro

Tutor:
Daniel Limón Marruedo
Profesor titular

Dep. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2016

Trabajo Fin de Grado: Implementación de controladores predictivos en Arduino

Autor: Alfonso García Navarro

Tutor: Daniel Limón Marruedo

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2016

El Secretario del Tribunal

# Agradecimientos

Este trabajo no hubiera sido posible sin la inestimable ayuda de mi tutor Daniel Limón, que pese a la persistencia de mis preguntas y consultas no dejó de responderme con celeridad y eficiencia, incluso en momentos en los que el trabajo le era primordial. Nos dedicó el tiempo que no tenía al adecuar su horario por nosotros durante un año.

En segundo lugar, y no por ello menos importante, a Pablo Krupa, que de manera desinteresada nos resolvió innumerables dudas a mí y a mis compañeros de *faena*, Ramón y Gonzalo, a los que también les agradezco su compañerismo y la ayuda mutua prestada.

A mi familia, amigos y compañeros.

De todo corazón, gracias y buena suerte en el futuro.

*Alfonso García Navarro*

*Sevilla, 2016*

# Resumen

El propósito principal de este trabajo es describir inicialmente los principios teóricos y matemáticos del Control Predictivo basado en Modelo (MPC) para luego implementarlo y simularlo en MATLAB®, acompañado de diferentes estrategias para la optimización del control.

Dichas estrategias son: la utilización del algoritmo FISTA (*A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems, Beck, 2009*) en la resolución del problema de optimización; la obtención de estados de equilibrio para la referencia mediante un segundo problema de optimización (*Steady-State Target Optimization*) y el uso de un observador de Luenberger para la estimación del estado actual. Estas y otras cuestiones se desarrollarán con detalle en el presente trabajo.

Con vistas a la validación del método empleado se simulará sobre la planta de cuatro tanques y dos actuadores y se compararán resultados con las diferentes estrategias aplicadas.

En una segunda parte del proyecto se implementará todo lo descrito anteriormente en un microcontrolador *Arduino® Due* para la posible aplicación sobre la planta real y con la posibilidad de adecuar las variables para el control de otro sistema.

# Abstract

The main purpose of this paper is to describe the theoretical and mathematical principles of Model Predictive Control (MPC) initially and then implement it in MATLAB® including several control strategies in order to make simulations.

These strategies will be: using FISTA algorithm (*A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems, Beck, 2009*) in the Optimization Problem; obtaining steady-state targets for set-point (*Steady-State Target Optimization*) and using a Luenberger observer in order to estimate current state. These and others issues will be expounded in detail in this work.

It will be simulated on the quadruple-tank with two actuators with the intention of validating the used methods and comparing different results.

In the second part of the project all stated above will be implemented in a *Arduino® Due* microcontroller in order to control the actual quadruple-tank and with the possibility of adjusting the parameters in order to use it in a different controllable system.

# Índice

# Índice de Tablas

# Índice de Figuras

# Notación

| | |
|---|---|
| MPC | Control Predictivo basado en Modelo |
| SSTO | *Steady-State Target Optimization* |
| PWM | Pulse-Width Modulation (modulación por ancho de pulsos) |
| SRAM | Static Random Access Memory (RAM estática) |
| UART | Transmisor-Receptor Asíncrono Universal |
| ISTA | Iterative Shrinkage Thresholding Algorithm |
| FISTA | Fast Iterative Shrinkage Thresholding Algorithm |
| LB | Lower Bounds |
| UB | Upper Bounds |
| c.t.p. | En casi todos los puntos |
| i.e. | Es decir |

# 1 INTRODUCCIÓN

*Inteligencia, dame el nombre exacto de las cosas.*

*- Juan Ramón Jiménez -*

El nacimiento del control moderno podría establecerse en la década de los 60 con el desarrollo del regulador lineal cuadrático (LQR) de manos de Rudolf E. Kalman. Este regulador minimiza una función cuadrática objetivo sin restricciones de los estados y entradas. Dicha ausencia de restricciones en su formulación, la característica no lineal de los sistemas reales y la falta de conocimiento en control óptimo en el ámbito general de la industria de proceso en ese momento, propició que el LQR apenas tuviera impacto en el desarrollo de la tecnología de control en dicha industria.

De esta manera, el MPC nació de la pura necesidad con las primeras aplicaciones exitosas de Richalet (1978) y Cutler y Ramaker (1980).

La idea principal de dichas estrategias fue la de usar un modelo dinámico del proceso para predecir el efecto de las futuras acciones de control, que se determinaron mediante la minimización del error predicho sujeto a unas restricciones operativas. En cada periodo de muestreo la optimización se repite con información actualizada del proceso.

Con el aumento de la capacidad computacional de los ordenadores del momento, surgieron estrategias más sofisticadas como la de Garcia y Morshedi (1986) en la que se utilizaba la programación cuadrática para resolver el problema de optimización.

De forma paralela al MPC surgió el control adaptativo para procesos monovariables formulados con modelos de entrada/salida. Un ejemplo sería el control de mínima varianza de Astron (1970).

Por último cabe señalar también la aparición del Control Predictivo Generalizado (GPC) a finales de la década de los 80.

La situación actual del MPC en su aplicación dentro de la industria abarca en su mayoría procesos multivariables, teniendo su protagonismo en la industria de proceso. A partir de finales de 1990 se hicieron numerosos estudios para posibles aplicaciones en diferentes campos. El éxito actual en la industria de proceso se debe tres razones principales:

- La incorporación de un modelo explícito del proceso en los cálculos permite al controlador tratar con todas las características importantes de la dinámica del proceso.

- La consideración del comportamiento del proceso a lo largo de un horizonte futuro permite tener en cuenta el efecto de las perturbaciones en alimentación y prealimentación, permitiendo al controlador conducir la salida a la trayectoria de referencia deseada.

- La consideración de restricciones en la fase de diseño del controlador evita en lo posible su violación, resultando un control más preciso en torno al punto óptimo de operación. La inclusión de restricciones es quizás la característica que más distingue al MPC respecto a otras metodologías.

# 2 DESCRIPCIÓN DEL PROYECTO

*Somos como enanos a los hombros de gigantes. Podemos ver más, y más lejos que ellos, no por la agudeza de nuestra vista ni por la altura de nuestro cuerpo, sino porque somos levantados por su gran altura.*

*Bernardo de Chartres*

En este apartado se describirán los principios teóricos y matemáticos del Control Predictivo basado en Modelo (MPC) así como una descripción del sistema de cuatro tanques y sus modelos que usaremos para simular y comparar las diferentes estrategias y componentes del bucle completo.

De forma breve se comentará también la plataforma en la que se implementará el resultado final, el microcontrolador *Arduino® Due* y sus posibles aplicaciones en el mundo real.

## 2.1. Control Predictivo basado en Modelo

Como se ha visto anteriormente el MPC tiene sus raíces en el control óptimo. El concepto básico del MPC es usar un modelo dinámico para predecir el comportamiento de un sistema y optimizar dicha predicción para determinar la mejor decisión en el tiempo adecuado. Los modelos son por lo tanto la base de toda forma de MPC. Debido a que toda acción de control óptimo depende del estado inicial del sistema dinámico, un segundo concepto básico en el MPC es usar las mediciones de acciones y estados pasados para determinar el estado inicial o actual más probable. Tanto el problema de regulación, en el que un modelo de predicción es usado para producir la acción de control óptima, como el problema de estimación, en el que la medición de datos pasados es usada para producir una estimación óptima del estado, incluyen modelos dinámicos y optimización.

En primer lugar presentaremos los modelos dinámicos de la forma más familiar, en forma de ecuaciones diferenciales:

$$\begin{aligned} \frac{dx}{dt} &= f(x,u,t) \\ y &= h(x,u,t) \\ x(t_0) &= x_0 \end{aligned} \tag{2.1}$$

En el que $x \in \mathbb{R}^n$ es el estado, $u \in \mathbb{R}^m$ es la entrada, $y \in \mathbb{R}^p$ es la salida y $t \in \mathbb{R}$ es el tiempo. Usamos $\mathbb{R}^n$ para denotar el conjunto de n-vectores reales. La condición inicial especifica el valor del estado $x$ en el instante $t = t_0$ , y buscamos una solución a la ecuación diferencial para tiempos mayores que $t_0$ , $t \in \mathbb{R}_{\geq t_0}$. Generalmente se define el instante inicial en 0, con su correspondiente condición inicial, en este caso $t \in \mathbb{R}_{\geq 0}$.

El modelo lineal más generalmente usado es el expresado en espacio de estados:

$$\begin{aligned} \frac{dx}{dt} &= Ax + Bu \\ y &= Cx + Du \\ x(0) &= x_0 \end{aligned} \tag{2.2}$$

En el cual $A \in \mathbb{R}^{n \times n}$ es la matriz de transición de estado, $B \in \mathbb{R}^{n \times m}$ es la matriz de entrada, $C \in \mathbb{R}^{p \times n}$ es la matriz de salida y $D \in \mathbb{R}^{p \times m}$ permite una relación directa entre $u$ y $y$. En muchas aplicaciones $D = 0$ como en nuestro caso. Cabe también señalar que $A$, $B$, $C$ y $D$ son invariantes en el tiempo, pudiendo no serlo en otras aplicaciones.

Cabe señalar en este punto que para la obtención de las matrices $A, B, C$ y $D$ para nuestro sistema a usar, el cual se mostrará más adelante, usaremos un punto de linealización del sistema. Definiremos a su vez las llamadas **variables incrementales**, las cuales estarán centradas en dicho punto de linealización. Así, si $X_f$ y $U_f$ representan los puntos de funcionamiento, las variables incrementales serán:

$$x = X - X_f$$

$$u = U - U_f$$

Donde $X$ y $U$ son las variables absolutas, i.e., el valor real dentro del sistema. Para el resto del documento (exceptuando el simulador no lineal, que será descrito a continuación) haremos uso siempre de dichas variables incrementales, centradas en el punto de linealización.

Una de las principales motivaciones para la utilización de modelos lineales para aproximar los sistemas físicos es la simplicidad de solución y análisis de dichos modelos.

### 2.1.1 Modelo en tiempo discreto

Para lo que nos compete definiremos el sistema de forma discreta.

Los modelos en tiempo discreto son normalmente convenientes si el sistema de interés se muestrea en tiempos discretos, i.e., sistemas de control implementados en computadoras digitales, como el *Arduino*®. Si el tiempo de muestreo es elegido de forma apropiada, el comportamiento entre las muestras puede ser ignorado con la suficiente seguridad de que el modelo describirá exclusivamente el comportamiento en los instantes muestreados.

Nuestro modelo será dimensionalmente finito, lineal, invariante en el tiempo y discreto:

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) \\ y(k) &= Cx(k) + Du(k) \\ x(0) &= x_0 \end{aligned} \tag{2.3}$$

En el cual $k \in \mathbb{Z}_{\geq 0}$ es un entero no negativo que denota el número de muestra, que está relacionada con el tiempo por $t = k \cdot T_m$ donde $T_m$ es el tiempo de muestreo. Usamos $\mathbb{Z}$ para denotar el conjunto de números enteros y $\mathbb{Z}_{\geq 0}$ para el conjunto de enteros no negativos. El modelo lineal en tiempo discreto es por lo tanto una relación de recurrencia lineal.

Es conveniente reescribir el índice temporal como un subíndice:

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k \\ y_k &= Cx_k + Du_k \end{aligned} \tag{2.4}$$

$$x_0 \ dado$$

Pero para reducir aún más la complejidad notacional nosotros escribiremos (2.3) en este texto de la siguiente manera:

$$\begin{gathered} x^+ = Ax + Bu \\ y = Cx + Du \\ x(0) = x_0 \end{gathered} \tag{2.5}$$

Donde el superíndice $^+$ significa el estado en el siguiente instante de muestreo. El modelo lineal en tiempo discreto es el más conveniente al presentar las ideas y conceptos del MPC de la forma matemática más simple posible.

### 2.1.2 Restricciones

Las variables manipulables (posición de una válvula, voltaje, par de fuerzas, etc.) de la gran mayoría de sistemas físicos están delimitadas. Nosotros incluimos estos límites mediante inecuaciones lineales:

$$Eu_k \leq e \qquad k \in \mathbb{Z}_{\geq 0} \tag{2.6}$$

Donde

$$E = \begin{bmatrix} I \\ -I \end{bmatrix} \qquad e = \begin{bmatrix} \overline{u} \\ -\underline{u} \end{bmatrix} \tag{2.7}$$

Son elegidos para describir límites simples como

$$\underline{u} \leq u_k \leq \overline{u} \qquad k \in \mathbb{Z}_{\geq 0} \tag{2.8}$$

Generalmente queremos imponer límites en estados o salidas por razones de seguridad, operabilidad, calidad, etc. Cuando consideramos sistemas no lineales, el análisis del controlador no es muy simplificado manteniendo estas inecuaciones lineales, por lo que generalizamos los límites mediante conjuntos:

$$x_k \in X \qquad u_k \in U \qquad k \in \mathbb{Z}_{\geq 0} \tag{2.9}$$

Debemos tener en cuenta una distinción general entre restricciones en la entrada y restricciones en la salida o estados. Las restricciones en la entrada generalmente representan límites *físicos*. En estos casos, si el controlador no respeta dichas restricciones, el sistema físico las hace cumplir. En cambio, las restricciones en las salidas o estados son normalmente *deseables*. Su cumplimiento puede no ser posible dependiendo de las perturbaciones que afectan al sistema. Es normalmente tarea de un MPC el determinar en tiempo real si las restricciones de las salidas o estados son realizables o no, y relajarlas de alguna manera satisfactoria si se da el caso. Como veremos más adelante, estas consideraciones llevan a implementaciones dentro del MPC para la obtención de resultados satisfactorios u óptimos si estas restricciones no pueden cumplirse.

### 2.1.3 Sistemas deterministas y estocásticos

Si examinamos las mediciones de un proceso físico y complejo, las fluctuaciones en los resultados están inevitablemente presentes tal y como se representa en la siguiente figura.

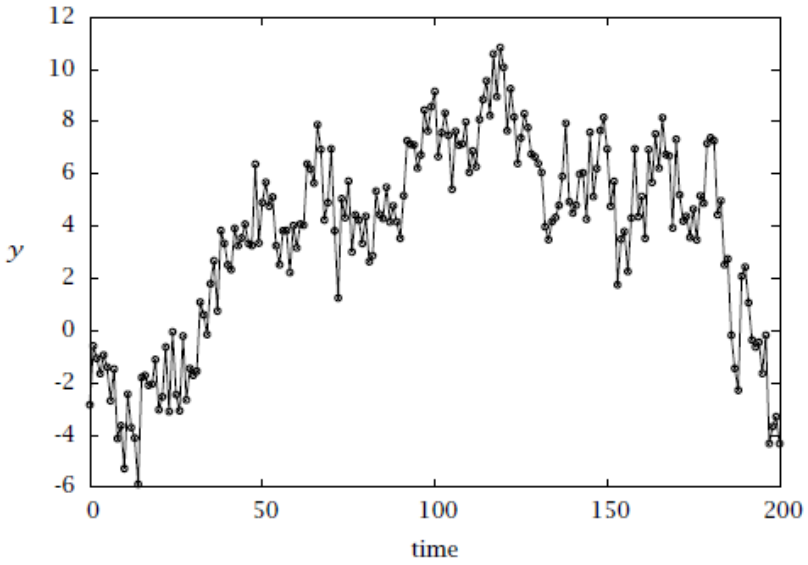



Figura 2-1. Salida de un sistema estocástico respecto al tiempo

Estas fluctuaciones pueden ser causadas por el comportamiento aleatorio de las partículas del sistema, por su interacción con el ambiente o por el ruido interno en el sistema o en el propio aparato de medida. Cuanto menor sea el sistema, mayor será la influencia de las fluctuaciones.

Los clásicos problemas del control suelen ser el modelado, monitorización y control de sistemas macroscópicos, i.e., no vamos a considerar sistemas compuestos de pequeñas cantidades de partículas para lo que nos compete en este documento, pero si uno examina las mediciones de cualquier proceso controlado de interés, sin importar lo *macroscópico* que sea, verá que la respuesta será parecida a la de la Figura 2-1. Si es importante tener en cuenta las fluctuaciones es necesario definir un modelo estocástico.

Como hemos dicho anteriormente, entre las posibles causas de estas perturbaciones se encuentra el *ruido* interno del aparato de medida y las interacciones con el ambiente no modeladas que afectan al estado del sistema. La manera más simple de representar estas dos posibles fuentes de perturbaciones en un modelo estocástico es un modelo lineal más unas variables aleatorias.

$$\begin{aligned} x^+ &= Ax + Bu + w \\ y &= Cx + Du + d \\ x(0) &= x_0 \end{aligned} \tag{2.10}$$

La variable $w \in \mathbb{R}^n$ es la variable aleatoria actuando en la transición de estados, $d \in \mathbb{R}^p$ es la variable aleatoria actuando en la salida medida y $x_0$ el estado inicial pudiendo también ser aleatorio. La variable aleatoria $d$ es usada para modelar el ruido de medida y $w$ para las perturbaciones en el sistema.

## 2.2 Introducción al regulador MPC

Comenzaremos diseñando un controlador para llevar un sistema lineal y determinista al origen. El modelo del sistema sería:

$$\begin{aligned} x^+ &= Ax + Bu \\ y &= Cx \end{aligned} \tag{2.11}$$

Inicialmente, asumiremos que el estado es medido, i.e., $C = I$. Del anterior punto sabemos que esto no es del todo cierto, trataremos la estimación de estado para la medida de la salida en próximos capítulos. Usando este modelo podemos predecir cómo evolucionará el estado dado para cualquier conjunto de entradas que

consideremos. Tomaremos $N$ instantes en el futuro y recogeremos la secuencia de entrada en $\mathbf{u}$

$$\mathbf{u} = \{u(o), u(1), \dots u(N-1)\} \tag{2.12}$$

A continuación definimos una función objetivo $V(\cdot)$ para medir la desviación de la trayectoria de $x(k), u(k)$ al cero mediante la suma de los cuadrados con diferentes *pesos*.

$$V(x(o), \mathbf{u}) = \frac{1}{2}\sum_{k=0}^{N-1} [x_k' Q\, x_k + u_k'\, R\, u_k] + \frac{1}{2} x_N'\, P_f\, x_N \tag{2.13}$$

Sujeto a

$$x^+ = Ax + Bu$$

La función objetivo depende del vector de entrada y del vector de estados. El estado inicial está disponible gracias a la primera medición. El resto de la trayectoria de estado, $x_k, k = 1, \dots, N$, es determinado por el modelo y el vector de entrada $\mathbf{u}$. Por lo que queda demostrada la dependencia explícita de la función objetivo con el vector de entrada y el estado inicial. Los parámetros de ajuste en el controlador son las matrices $Q$ y $R$. La diferencia entre el estado final y el origen tiene una matriz de peso diferente, $P_f$, que, en el caso que este documento nos ocupa, tomaremos $P_f = 0$.

Valores grandes de $Q$ en comparación con $R$ reflejan un intento de dirigir el estado al origen de manera rápida a expensas de grandes acciones de control. Penalizando la acción de control mediante grandes valores de $R$ respecto a $Q$ es la forma de reducir la acción de control y retrasar el ritmo con el que el estado se acerca al origen. Eligiendo valores apropiados de $Q$ y $R$ (i.e., sintonizar) no es siempre trivial, y esta dificultad es uno de los retos a los que se enfrentan los técnicos de la industria del MPC.

Formulamos ahora el siguiente problema de control óptimo LQ

$$min\, V(x(0), \mathbf{u}) \tag{2.14}$$

Las matrices $Q$ y $R$ serán diagonales y definidas positivas para el resto del documento. Esta suposición garantiza que la solución al problema de control óptimo existe y es única.

### 2.2.1 Controlabilidad

Un sistema es *controlable* si, para cualquier par de estados $x, z$ en el espacio de estados, $z$ es alcanzable en un tiempo finito desde $x$ (Sontag, 1998). Un sistema lineal de tiempo discreto $x^+ = Ax + Bu$ es por lo tanto controlable si existe un instante finito $N$ y una secuencia de entrada

$$\{u(0), u(1), \dots u(N-1)\}$$

que puede transferir al sistema desde cualquier $x$ a cualquier $z$, la cual

$$z = A^N x + [B \quad AB \quad \cdots \quad A^{N-1}B] \begin{bmatrix} u(N-1) \\ u(n-2) \\ \vdots \\ u(0) \end{bmatrix} \tag{2.15}$$

Podemos simplificar esta condición observando que las potencias de la matriz $A^k$ para $k \geq n$ se pueden expresar como combinaciones lineales de las potencias desde 0 hasta $n-1$. Por lo tanto el rango de la matriz $[B \quad AB \quad \cdots \quad A^{N-1}B]$ para $N \geq n$ es el mismo que $[B \quad AB \quad \cdots \quad A^{n-1}B]$. En otras palabras, para un sistema lineal sin restricciones, si no podemos alcanzar $z$ en $n$ movimientos, no podremos alcanzar $z$ en cualquier número de movimientos. La cuestión de *controlabilidad* de un sistema lineal invariante en el tiempo es por lo tanto una cuestión de *existencia* de soluciones a ecuaciones lineales para un miembro de la ecuación arbitrario.

$$z - A^n x = [B \quad AB \quad \cdots \quad A^{n-1}B] \begin{bmatrix} u(n-1) \\ u(n-2) \\ \vdots \\ u(0) \end{bmatrix} \tag{2.16}$$

La matriz que aparece en esta ecuación recibe el nombre de *matriz de controlabilidad C*

$$C = [B \quad AB \quad \cdots \quad A^{n-1}B] \tag{2.17}$$

De mano de los fundamentos del álgebra lineal, sabemos que existe una solución para todo miembro de la ecuación si y solo si las filas de la matriz de controlabilidad ($n \times nm$) son linealmente independientes. Por lo tanto, el sistema $(A, B)$ es controlable si y solo si

$$rang(C) = n \tag{2.18}$$

## 2.3 Planta de los 4 Tanques

Aunque en primera instancia la idea de trabajo es diseñar un controlador MPC ajustable a cualquier sistema, a lo largo del proyecto, usaremos el modelo de la planta de cuatro tanques existente en los Laboratorios del Departamento de Automática para probar y simular los distintos métodos y valores. Esta planta resulta muy conveniente pues ofrece un sistema lo suficientemente complejo ($n = 4$, $m = 2$, ceros de fase no mínima) como para validar la eficiencia del controlador ante sistemas reales de la industria de procesos.

A continuación se describirá la planta y su modelo.

### 2.3.1 Descripción de la planta de los cuatro tanques

Es una implementación de la planta propuesta por Karl H. Johansson en el artículo titulado "*The quadruple-tankprocess",* publicado en la revista *IEEE Transactionson Control Systems Technology, vol 8* (2000) [1]. Esta planta ha sido ampliamente usada como planta de laboratorio debido a su interés como problema de control y su sencillez de construcción y uso.

En la Figura 2-2, que se muestra a continuación, se ilustra el esquema de la planta de Johansson. Ésta consta de 4 depósitos, 2 inferiores (tanques 1 y 2) y 2 superiores (tanques 3 y 4) que desaguan en los tanques inferiores. Los tanques se llenan con 2 bombas que impulsan sendos caudales $q_a$ y $q_b$ desde el depósito colector situado en la parte baja de la planta. Estos caudales entran en sendas válvulas de 3 vías que dividen el caudal entre 2 ramas, de forma que una fracción (denominadas $\gamma_a$ y $\gamma_b$) se envía por una de las ramas y el resto por la otra. Las fracciones $\gamma_a$ y $\gamma_b$ se indican en tanto por uno y se fijan manualmente mediante la apertura de dichas válvulas.

De esta forma, el caudal $\gamma_a q_a$ entra en el tanque 1 y el caudal $(1 - \gamma_a)q_a$ entrará en el tanque 4. Del mismo modo, el caudal $\gamma_b q_b$ entra en el tanque 2 y el caudal $(1 - \gamma_b)q_b$ entrará en el tanque 3. El tanque 3 se descarga sobre el tanque 1 y este sobre el depósito colector. El tanque 4 se descarga sobre el 2 y este en el tanque colector.

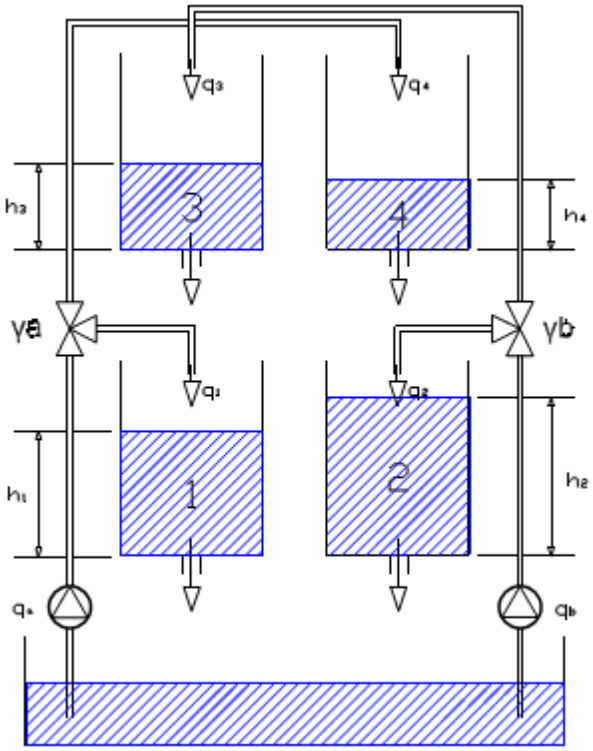



Figura 2-2. Esquema de la planta de Johansson

Esta planta es multivariable pues presenta dos variables manipulables $q_a$ y $q_b$. Las variables que se controlarán serán el nivel de líquido en los depósitos inferiores. La dinámica de esta planta puede presentar ceros de transmisión según los valores de la apertura de las válvulas de 3 vías, $\gamma_a$ y $\gamma_b$ (véase el artículo de Johansson). Asimismo, la planta presenta una dinámica no lineal y se describe bien mediante un modelo de orden 4 basado en primeros principios. Además la planta presenta restricciones en las actuaciones (caudales máximos y mínimos) y en las variables del proceso (niveles máximos y mínimos en los depósitos).

### 2.3.2 Modelo de la planta de los cuatro tanques

El modelo de la planta de Johansson se puede derivar de primeros principios. Este modelo viene dado por las siguientes ecuaciones diferenciales

$$\begin{aligned} A\frac{dh_1}{dt} &= -a_1\sqrt{2gh_1} + a_3\sqrt{2gh_3} + \gamma_a\frac{q_a}{3600} \\ A\frac{dh_2}{dt} &= -a_2\sqrt{2gh_2} + a_4\sqrt{2gh_4} + \gamma_b\frac{q_b}{3600} \\ A\frac{dh_3}{dt} &= -a_3\sqrt{2gh_3} + (1-\gamma_b)\frac{q_b}{3600} \\ A\frac{dh_4}{dt} &= -a_4\sqrt{2gh_4} + (1-\gamma_a)\frac{q_a}{3600} \end{aligned} \tag{2.19}$$

En las que $h_i$ denota el nivel de líquido en el tanque $i$. Este nivel se mide en metros. Los caudales de cada rama medidos en metros cúbicos por hora se denotan $q_a$ y $q_b$ y son variables manipulables.

El parámetro $a_i$ es la sección equivalente del orificio de descarga del depósito $i$ y la sección de todos los tanques es igual y se denota por $A$. Las secciones se miden en metros cuadrados. Los parámetros $\gamma_a$ y $\gamma_b$

indican la apertura de la válvula de 3 vías. La unidad de tiempo del modelo es segundos.

La actualización del valor de las variables manipulables que se aplicarán y la actualización de las medidas de los sensores se realiza cada 5 segundos. Por lo que $T_m = 5$.

Este modelo de primeros principios se considera un modelo ideal pues no describe dinámicas que presenta la planta real tales como la evolución de las presiones en las tuberías o los caudales de alimentación a cada uno de los tanques. Tampoco describe el efecto de los vórtices en la descarga de los tanques ni la dinámica y ruido que introduce la instrumentación. Por lo tanto existe un error entre evolución de los niveles prevista por el modelo y la de los niveles de la planta real.

La ecuación linealizada en el espacio de estados es:

$$\frac{dx}{dt} = \begin{bmatrix} -\frac{1}{T_1} & 0 & \frac{A}{AT_3} & 0 \\ 0 & -\frac{1}{T_2} & 0 & \frac{A}{AT_4} \\ 0 & 0 & -\frac{1}{T_3} & 0 \\ 0 & 0 & 0 & -\frac{1}{T_4} \end{bmatrix} x + \begin{bmatrix} \frac{\gamma_1 k_1}{A} & 0 \\ 0 & \frac{\gamma_2 k_2}{A} \\ 0 & \frac{(1-\gamma_2)k_2}{A} \\ \frac{(1-\gamma_1)k_1}{A} & 0 \end{bmatrix} u \tag{2.20}$$

$$y = \begin{bmatrix} k_c & 0 & 0 & 0 \\ 0 & k_c & 0 & 0 \end{bmatrix} x$$

Donde las constantes de tiempo son

$$T_i = \frac{A}{a_i}\sqrt{\frac{2h_i^0}{g}} \quad , \quad i = 1, \dots, 4 \tag{2.21}$$

Los parámetros $k_{1,2}$ y $k_c$ son para transformar, respectivamente, voltaje a caudal para la apertura de las válvulas y altura a voltaje para las salidas.

A nuestra disposición tendremos un código MATLAB® que nos proporciona el modelo lineal de la planta. Calcula las matrices $A$, $B$, $C$, y $D$ para un punto de funcionamiento concreto y un tiempo de muestreo dado.

**Código 2.1** Código MATLAB® : Modelo lineal de la planta

```
%% Funcion [A, B, C, D] = modelo4tanques ( Xf, Tm)
% Calcula las matrices A, B, C y D del modelo en ecuaciones de estado de la
% planta de cuatro tanques para un punto de funcionamiento dado por Xf
% y para un tiempo de muestreo Tm

function [A, B, C, D] = modelo4tanques( Xf, Tm)
% Constantes
a = 0.03;
g=9.81;
% Calculo de los elementos distintos de 0 de la matriz A
A11 = 1 + Tm*( (-1.3104e-4*g)/(a*sqrt(2*g*Xf(1))));
A13 = Tm * ((9.2673e-5*g) / (a*sqrt(2*g*Xf(3))));
A22 = 1 + Tm*( (-1.5074e-4*g) / (a*sqrt(2*g*Xf(2))));
A24 = Tm * ((8.8164e-5*g) / (a*sqrt(2*g*Xf(4))));
A33 = 1 + Tm*( (-9.2673e-5*g) / (a*sqrt(2*g*Xf(3))));
A44 = 1 + Tm*( (-8.8164e-5*g) / (a*sqrt(2*g*Xf(4))));
% Calculo de los elementos distintos de 0 de la matriz B
B11 = (0.3*Tm) / (3600*a);
B22 = (0.4*Tm) / (3600*a);
```

```
B32 = (0.6*Tm) / (3600*a);
B41 = (0.7*Tm) / (3600*a);
% Creacion de las matrices A y B
A = [A11  0    A13   0  ;
     0    A22  0     A24;
     0    0    A33   0  ;
     0    0    0     A44];
B = [B11  0  ;
     0    B22;
     0    B32;
     B41  0  ];
C = [1 0 0 0;
     0 1 0 0];
D = [0 0;
     0 0];
end
```

Para el cálculo del punto de funcionamiento también disponemos de un código MATLAB® que lo calcula para cualquier combinación de caudales.

Las restricciones de la planta son $0.2 < x < 1.2$ y $0 < u < 3$ en valores absolutos. Esto habrá que corregirlo con el punto de funcionamiento pues el MPC trabaja con variables incrementales, centradas en el punto de funcionamiento. Más adelante concretaremos los valores usados.

**Código 2.2** Código MATLAB® : Calculo del punto de funcionamiento de la planta

```
%% Funcion Xf = puntoFunc(Q)
% Calcula el punto de funcionamiento de la planta de cuatro tanques (Xf) a
% partir de un vector de caudal (Q).

function Xf = puntoFunc(Q)
% Declaracion del vector Xf
Xf = zeros(4,1);
% Calculo de sus componentes
Xf(3) = (( ( (0.6*Q(2))/3600 ) / ( 9.2673e-5 ) )^2) / (2*9.81);
Xf(4) = (( ( (0.7*Q(1))/3600 ) / ( 8.8164e-5 ) )^2) / (2*9.81);
Xf(1) = (( ( (0.3*Q(1))/3600 + (9.2673e-5)*sqrt(2*9.81*Xf(3)) ) / ( 1.3104e-4
) )^2) / (2*9.81);
Xf(2) = (( ( (0.4*Q(2))/3600 + (8.8164e-5)*sqrt(2*9.81*Xf(4)) ) / ( 1.5074e-4
) )^2) / (2*9.81);
end
```

### 2.3.3 Simulador no lineal de la planta de los cuatro tanques

En el ámbito del control automático, es bien sabido que un modelo lineal no siempre reflejará la respuesta real de un sistema. Es por ello que contamos también para este trabajo con un simulador no lineal de la planta, el cual se ajustará de manera más apropiada a la respuesta real del sistema de cuatro tanques.

Este simulador se basa en el método de Euler el cual es un procedimiento de integración numérica para resolver ecuaciones diferenciales a partir de un valor inicial dado. De manera general el método se puede describir como una división de los intervalos que va de $x_0$ a $x_f$ en $n$ subintervalos de ancho $h$, i.e.:

$$h = \frac{x_f - x_0}{n} \tag{2.22}$$

De manera que se obtiene un conjunto discreto de $n+1$ puntos: $x_0, x_1, x_2, \cdots, x_n$ del intervalo de interés $[x_0, x_f]$ . Para cualquiera de estos puntos se cumple que:

$$x_i = x_0 + ih \qquad 0 \leq i \leq n \tag{2.23}$$

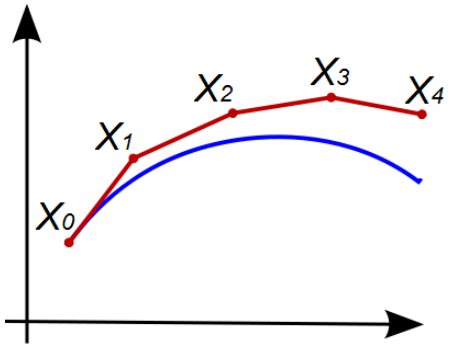



Figura 2-3. Ilustración del método de Euler

La respuesta real se representa en azul y la aproximación poligonal del método de Euler en rojo.

La condición inicial $y(x_0) = y_0$ representa el punto $(x_0, y_0)$ por donde pasa la curva solución de la respuesta real. Teniendo ese punto, se puede evaluar la primera derivada de dicha curva en ese punto. Con esta información ya podemos trazar una recta, aquella que pasa por $(x_0, y_0)$ y de pendiente el valor de la anterior derivada. Esta recta aproxima la respuesta real en una vecindad de $x_0$.

A continuación, calculamos la pendiente de la recta:

$$\frac{y_1 - y_0}{x_1 - x_0} = f(x_0, y_0) \tag{2.24}$$

Y llegamos a la expresión:

$$y_1 = y_0 + (x_1 - x_0) f(x_0, y_0) = y_0 + hf(x_0, y_0) \tag{2.25}$$

Es evidente que la ordenada $y_1$ calculada de esta manera no es igual al valor que tendría en la curva real de la respuesta, pues existe un pequeño error. Sin embargo, el valor $y_1$ sirve para que se aproxime la siguiente derivada de la respuesta en el punto $(x_1, y_1)$ y repetir el procedimiento anterior a fin de generar la sucesión de aproximación siguiente:

$$\begin{aligned} y_1 &= y_0 + hf(x_0, y_0) \\ y_2 &= y_1 + hf(x_1, y_1) \\ &\vdots \\ y_{i+1} &= y_i + hf(x_i, y_i) \\ &\vdots \\ y_n &= y_{n-1} + hf(x_{n-1}, y_{n-1}) \end{aligned} \tag{2.26}$$

De nuestro interés es la última expresión de (2.26) pues es la que se computerizará para calcular la respuesta de cada uno de los tanques de la planta.

**Código 2.3** Código MATLAB® : Simulador no lineal de la planta

```
%% Funcion Xk1 = simula_NL_4Tanques(Xk, Uk, Tm, h)
% Realiza la simulacion del sistema de cuatro tanques durante un tiempo de
% muestreo.
% Parametros
% Xk : Estado actual del sistema (alturas absolutas). Vector de dimension 4
% Uk : Actuacion que se le aplica (caudales absolutos). Vector de dimension 2
    % Tm : Tiempo de muestreo del sistema
    % h : Tiempo de integracion. Tomar uno que sea divisor de Tm y mucho
    %     mas pequeño que este. Por ejemplo: Tm = 5; h = 0.01;
% Devuelve el estado en el tiempo de muestreo siguiente, Xk1.
function Xk1 = simula_NL_4Tanques(Xk, Uk, Tm, h)

  for i=1:(Tm/h)
        Xk1(1) = Xk(1) + h*((-1.3104e-4*sqrt(2*9.81*Xk(1))+9.2673e-
5*sqrt(2*9.81*Xk(3))+(0.3/3600)*Uk(1))/0.03);
        Xk1(2) = Xk(2) + h*((-1.5074e-4*sqrt(2*9.81*Xk(2))+8.8164e-
5*sqrt(2*9.81*Xk(4))+(0.4/3600)*Uk(2))/0.03);
        Xk1(3) = Xk(3) + h*((-9.2673e-
5*sqrt(2*9.81*Xk(3))+(0.6/3600)*Uk(2))/0.03);
        Xk1(4) = Xk(4) + h*((-8.8164e-
5*sqrt(2*9.81*Xk(4))+(0.7/3600)*Uk(1))/0.03);
        for j=1:4
            if Xk1(j)<0
                Xk1(j) = 0;
            end
        end
        Xk = Xk1;
  end

end
```

Tanto el estado como la actuación funcionan como valores absolutos, a diferencia del MPC que funciona en valores incrementales, por lo que el punto de linieali̇zación debe ser sustraído y sumado cuando corresponda, cumpliendo con los límites descritos en la sección 2.3.2.

$h$ representa el tiempo de integración descrito anteriormente. Éste debe ser un valor divisor del tiempo de muestreo de forma que la $n$ del bucle resultante de la ecuación (2.26) sea un número entero de bucles. Mientras más pequeño es $h$ más exacto es el proceso de integración y más exacta la aproximación. Para nuestro caso tomaremos $h = 0.01$.

# 3 OPTIMIZACIÓN

En el siguiente capítulo se detallará el problema de la minimización de la función objetivo mediante la resolución de un problema de optimización cuadrática.

## 3.1 Minimización de la función objetivo

Como se ha dicho anteriormente, esta función la realiza el bloque ***MPC***. El objetivo y motivación de este cálculo se expuso en la **Sección 2.2**. A modo de recordatorio mostramos de nuevo la función objetivo[1].

$$\min_{u} \quad V(x(o),\mathbf{u}) = \frac{1}{2}\sum_{k=0}^{N-1}[x_k' Q\, x_k + u_k' R\, u_k] \tag{3.1}$$

Llegados a este punto, es necesario una descripción de cómo se va a abordar el problema de optimización.

### 3.1.1 El problema de optimización cuadrática

La definición de optimización matemática podría ser la búsqueda del mejor elemento o conjunto de elementos, sujeta a posibles restricciones, entre un conjunto de elementos disponibles. En concreto, nosotros afrontaremos el problema de optimización llamado **programación cuadrática**.

La programación cuadrática es un tipo especial de optimización matemática, la cual consiste en minimizar o maximizar una función cuadrática dada, con variables sujetas a restricciones lineales. Forma parte de la programación no lineal.

La formulación del problema es la siguiente.

$$\begin{aligned} &minimizar \quad \frac{1}{2}x^T H x + f^T x \\ &sujeto\ a \quad Ax \leq b \end{aligned} \tag{3.2}$$

Donde $f$ es un vector real de dimensión $n$; $H$ una matriz simétrica real de dimensiones $n \times n$; $A$ una matriz

[1] Para $P_f = 0$

real de dimensiones $m \times n$ y $b$ un vector real de dimensión $m$. Se dice entonces que el problema es de $n$ varialbes y con $m$ restricciones y su objetivo es encontrar un vector $x$ de dimensión $n$ que cumpla lo expuesto en (3.2). De ahora en adelante, para nosotros la matriz $H$ será una matriz diagonal y definida positiva.

Para exponer de forma más detallada el método que seguiremos para la resolución del problema de optimización cuadrática, iremos exponiendo ejemplos simples y los principios matemáticos que usaremos para su solución, hasta llegar al problema con la notación deseada.

En principio es necesario saber que dada la función cuadrática $\frac{1}{2}dx^2 + fx + c$ ésta será convexa si $d > 0$. Una función convexa es aquella que cumple

$$f''(x) \geq 0 \quad \forall x \tag{3.3}$$

y de manera gráfica en la Figura 3-1:

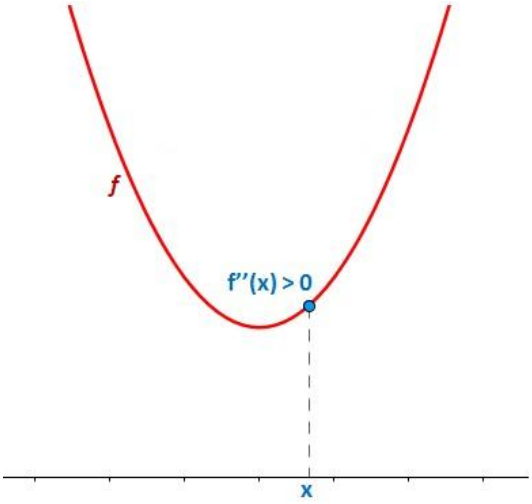



Figura 3-1. Función convexa en x

Planteamos entonces el siguiente ejemplo

---

**Ejemplo 3–1.** *Sea z un número real, calcular la solución del problema*

$$\begin{aligned} \min_z \quad & \frac{1}{2}dz^2 + fz \\ s.a. \quad & |z| \leq a \end{aligned}$$

*Siendo* $a > 0$. *Considerar tres casos:* $d > 0$, $d = 0$ *y* $d < 0$.

---

Para una primera simplificación del problema, suponemos $d, f, a = 1$. Por lo que el problema quedaría

$$\begin{aligned} & \frac{1}{2}z^2 + z \\ & |z| \leq 1 \end{aligned} \tag{3.4}$$

A continuación reescribimos la función con un parámetro $z_0$ para poder colocar el mínimo según su valor: $\frac{1}{2}(z - z_0)^2 \rightarrow \frac{1}{2}z^2 - z_0 z + \frac{1}{2}z_0^2$. Cabe señalar que esta simplificación añade un término a la función original modificando su valor en el eje $y$ el cual no nos afectará al problema, pues nosotros buscamos para que valor de $z$ ocurre ese mínimo, y no el valor de la función. De este modo, si por ejemplo tomamos $z_0 = -1$ la función quedaría de la forma $\frac{1}{2}(z + 1)^2$ y sería de la forma representada en azul en la Figura 3-2.

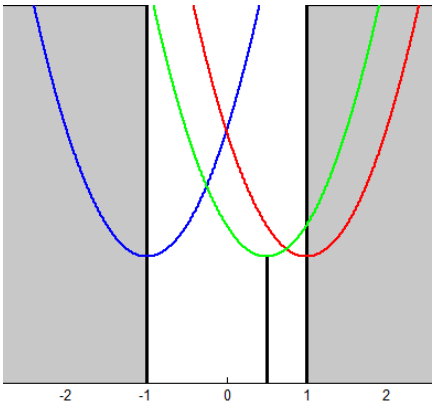



Figura 3-2. Mínimos en las funciones del Ejemplo 3–1.

En rojo y en verde se han representado los casos para $z_0 = 1$ y $z_0 = \frac{1}{2}$ respectivamente. La zona sombreada representa los límites pues como vimos en (3.4) $|z| \leq 1$. Todo lo anterior nos servirá a continuación.

Tomamos ahora $a$, $f$ y $d$ conocidos y seguimos la misma premisa haciendo $\frac{1}{2}(z - z_0)^2 d \rightarrow \frac{1}{2}dz^2 - dz_0 z + \frac{1}{2}z_0^2 d$. Igualando términos con el enunciado del **Ejemplo 3–1** es fácil ver que $f = -dz_0 \rightarrow z_0 = -\frac{f}{d}$ quedando la ecuación de la siguiente manera

$$\begin{aligned} \min_z \quad & \frac{1}{2}d\left(z + \frac{f}{d}\right)^2 \\ s.a. \quad & |z| \leq a \end{aligned} \tag{3.5}$$

Por lo tanto, como vimos anteriormente, se conoce que el mínimo de la función estará en $z_0 = -f/d$ delimitado por los límites $-a$ y $a$. Esto se ilustra en la Figura 3-3.

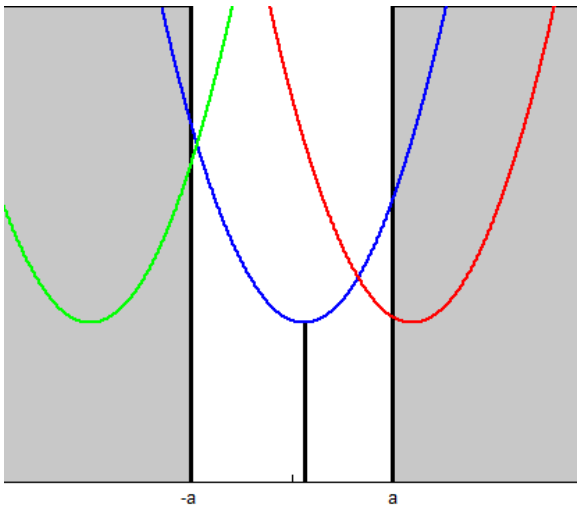



Figura 3-3. Mínimos para $a$, $f$ y $d$ conocidos

Como vemos en la figura, habrá valores de $f$ y $d$ para los que el mínimo de la función no pertenezca a los límites dados por el problema, siendo en esos casos los propios límites el punto donde se da el mínimo.

Vamos a estudiar ahora para qué valores ocurrirá lo anterior.

Tal y como nos pide el enunciado del problema, distinguimos 3 casos: $d > 0$, $d = 0$ y $d < 0$.

Caso 1. $d > 0$

Como acabamos de ver en la Figura 3-3 el mínimo de la función no tiene por qué coincidir con la solución al problema. Solo en el caso en el que se cumpla $|z_0| \leq a$, la solución será $z_0 = -f/d$. En cualquier otro caso la solución al problema será $a$ o $-a$. Es por ello que para el Caso 1 podemos escribir la solución de la siguiente manera.

$$z^* \quad \begin{cases} a & -\frac{f}{d} > a \\ -\frac{f}{d} & \left|\frac{f}{d}\right| \leq a \\ -a & -\frac{f}{d} < -a \end{cases} \tag{3.6}$$

Donde $z^*$ denota la solución del problema. La Figura 3-4 es una representación de la solución $z^*$.

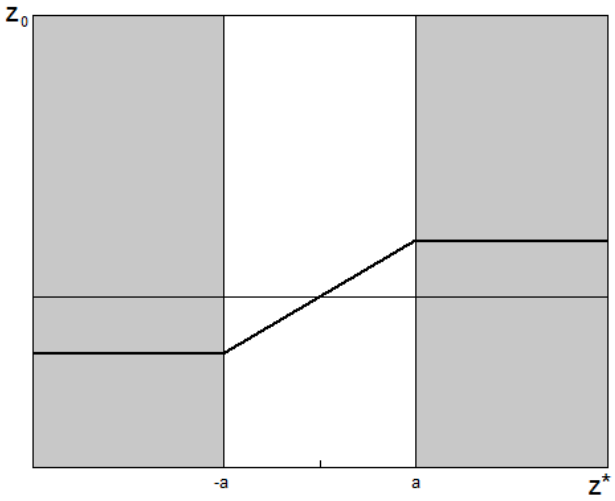



Figura 3-4. Valor de la solución $z^*$ respecto a $z_0$

Caso 2. $d = 0$

Para este caso el problema queda:

$$\begin{aligned} \min_z \quad & fz \\ s.a. \quad & |z| \leq a \end{aligned} \tag{3.7}$$

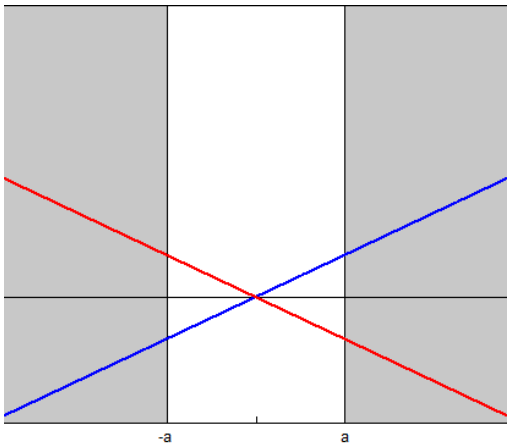



Figura 3-5. Representación de la función Caso 2

Como vemos en la representación de $fz$ este caso es trivial, pues para $d = 0$ las soluciones serán siempre los límites, $a$ para el caso en que $f$ sea negativa (ilustrado en rojo) y $-a$ para el caso contrario (azul).

Caso 3. $d < 0$

Este es el caso menos intuitivo pues, como vimos al declarar la ecuación (3.3), al ser $d$ menor que 0, la función resultante será cóncava. La siguiente figura facilita la comprensión.

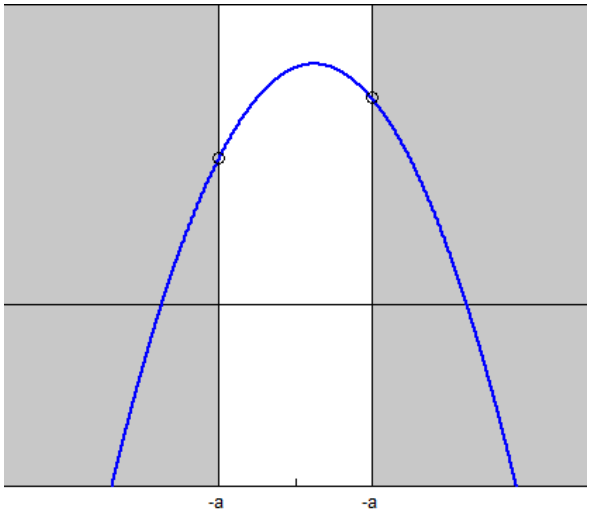



Figura 3-6. Representación para el caso no convexo

En este caso habría que evaluar la función en los extremos para saber cuál es el mínimo. Notar que, como ya dijimos en la **Subsección 3.1.1**, al ser nuestra matriz $H$ definida positiva, no nos encontraremos con casos no convexos.

---

**Ejemplo 3–2.** *Sea z un número real, calcular la solución del problema*

$$\begin{aligned} \min_{z} \quad & \frac{1}{2}dz^2 + fz \\ s.a \quad & a \leq z \leq b \end{aligned}$$

*Siendo* $a < b$.

---

Vamos a reescribir el problema de la siguiente manera. Restaremos $\frac{a+b}{2}$ a todos los elementos de la inecuación de la restricción. El problema quedaría sujeto a:

$$-\frac{(b-a)}{2} \leq z - \frac{a+b}{2} \leq \frac{b-a}{2}$$

Hacemos ahora el siguiente cambio de variable

$$y = z - \frac{a+b}{2} \rightarrow z = y + \frac{a+b}{2}$$

La función del problema quedaría entonces

$$\begin{gathered} \frac{1}{2}dz^2 + fz \rightarrow \frac{1}{2}\hat{d}y^2 + \hat{f}y \\ |y| \leq \hat{a} \end{gathered} \tag{3.8}$$

Donde $\hat{a} = \frac{b-a}{2}$.

Observando (3.8) vemos que el problema tiene ahora la misma forma que el del **Ejemplo 3–1**. Resolviendo de la misma manera y realizando el cambio de variable llegamos a la solución para $z^*$.

$$z^* \quad \begin{cases} a & -\frac{f}{d} < a \\ -\frac{f}{d} & a \le -\frac{f}{d} \le b \\ b & -\frac{f}{d} > b \end{cases} \tag{3.9}$$

**Ejemplo 3–3**. *Sea $z \in \mathbb{R}^n$ un vector de números reales, calcular la solución del problema*

$$\begin{aligned} \min_z \quad & \frac{1}{2} z^T D z + f^T z \\ s.a. \quad & |z_i| \le a_i, \quad i = 1, \dots, n \end{aligned}$$

*Siendo $a_i > 0$ y siendo $D \in \mathbb{R}^{n \times n}$ una matriz diagonal tal que $d_{ii} > 0$.*

La premisa que seguiremos será la misma, i.e., simplificar el problema hasta conseguir de nuevo la formulación del **Ejemplo 3–1**. Para ello usaremos un problema en el que $i = 2$.

$$\min_{z_1, z_2} \quad \frac{1}{2} z_1^2 d_1 + \frac{1}{2} z_2^2 d_2 + f_1 z_1 + f_2 z_2 \tag{3.10}$$

De forma matricial lo podemos escribir:

$$\frac{1}{2} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}^T \begin{bmatrix} d_1 & 0 \\ 0 & d_1 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} + \begin{bmatrix} f_1 & f_2 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} \tag{3.11}$$

Y los límites se pueden reescribir como

$$\begin{aligned} -a_1 \le z_1 \le a_1 \\ -a_2 \le z_2 \le a_2 \end{aligned} \tag{3.12}$$

En la Figura 3-7 vemos representado el punto a buscar en este problema.

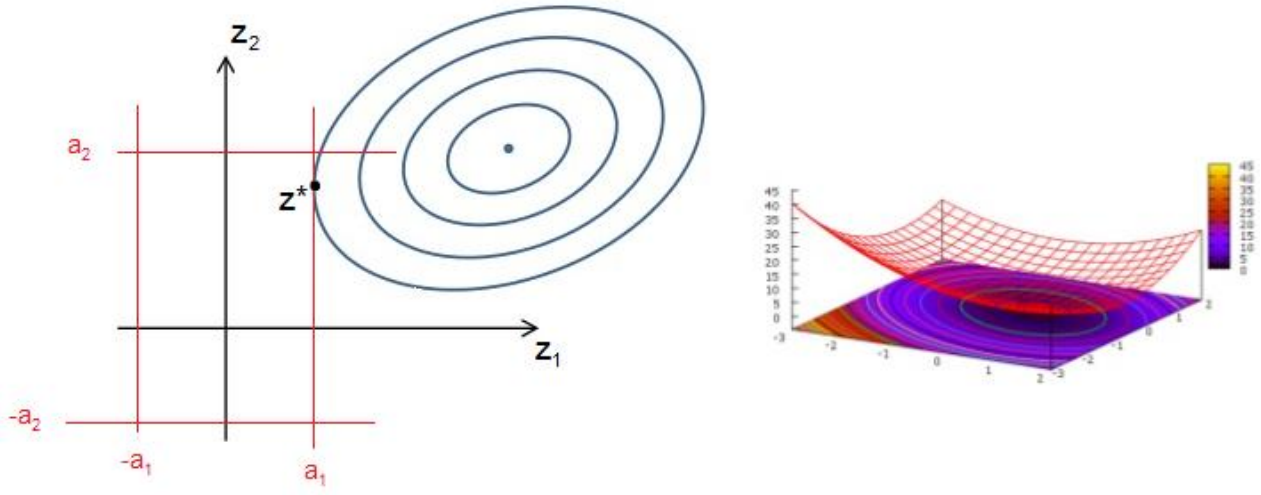



Figura 3-7. Minimización de una función cuadrática con restricciones cuadradas

Procederemos de la misma forma que lo hicimos en el **Ejemplo 3–1**. Declaramos el valor $z_0$ de forma que:

$$\frac{1}{2}(z-z_0)^T D(z-z_0) = \frac{1}{2}z^T Dz - z_0^T Dz + \frac{1}{2}z_0^T Dz_0 \tag{3.13}$$

Obviando el último término, pues como hemos dicho anteriormente solo afecta al valor de la función y no al punto donde ocurre, y haciendo el siguiente cambio de variable: $Dz_0 = -f \rightarrow z_0 = -D^{-1}f$ ya conocemos el mínimo. Volviendo a la ecuación (3.10) podemos ordenarla de la siguiente manera

$$\left(\frac{1}{2}z_1^2 d_1 + f_1 z\right) + \left(\frac{1}{2}z_2^2 d_2 + f_2 z_2\right) \tag{3.14}$$

Con sus respectivas restricciones $|z_1| \leq a_1$ y $|z_2| \leq a_2$, lo que lo convierte en dos problemas diferenciables con sus respectivos mínimos, a los que podemos aplicarles los mismos criterios descritos en los ejemplos anteriores.

$$z_1^* = F(d_1, f_1, a_1) \quad z_2^* = F(d_2, f_2, a_2) \tag{3.15}$$

Y así para cualquier $i$ dada.

De esta forma podemos ya implementar un código MATLAB® que, dado $d$, $f$ y $a$ calcule la saturación de la forma

$$\min(|x|, b) \cdot \frac{x}{|x|}$$

Ecuación que cumple con (3.9)

**Código 3.1** Código MATLAB® : Saturación de la $x$

```
function y=saturacion(d,f,a)
n=size(d,1);
y=zeros(1,n)';
for i=1:n
    z=-f(i,1)/d(i,i);
    y(i)=min(abs(z),a(i,1))*sign(z);
end
end
```

**Ejemplo 3–4**. *Sea $z \in \mathbb{R}^n$ un vector de números reales, calcular la solución del problema*

$$\min_z \quad \frac{1}{2}z^T Hz + f^T z$$
$$s.a. \quad |z_i| \leq a_i, \quad i = 1, \ldots, n$$

*Siendo $a_i > 0$. Considerar matriz H definida positiva.*

Para la resolución de este es necesario introducir otra subsección para el **algoritmo FISTA**.

#### 3.1.1.1 El algoritmo FISTA

El algoritmo FISTA (*Fast Iterative Shrinkage Thresholding Algorithm*), algoritmo rápido e iterativo de minimización mediante umbral según sus siglas, es un método de gradiente próximo usado para resolver problemas de optimización convexa para funciones no diferenciables de la forma

$$\min_{x \in \mathbb{R}^N} f_1(x) + f_2(x) + \cdots + f_{n-1}(x) + f_n(x) \tag{3.16}$$

Donde $f_1, f_2, \ldots, f_n$ son funciones convexas definidas desde $f: \mathbb{R}^N \rightarrow \mathbb{R}$ donde una o varias funiones pueden

ser no diferenciables [4]. Para ello realizaremos las siguientes modificaciones matemáticas del problema. Sea la función de coste:

$$f(z) = \frac{1}{2} z^T H z + f^T z \tag{3.17}$$

Y sea $L = \lambda_{max}(H)$, entonces aprovechando que $z^T H z \leq L z^T z$ para todo $z$, se va a derivar una aproximación local en torno a un punto $z_k$.

$$\begin{aligned} f(z_k + \Delta z) &= \frac{1}{2}(z_k + \Delta z)^T H (z_k + \Delta z) + f^T (z_k + \Delta z) \\ &= \frac{1}{2}\Delta z^T H \Delta z + z_k^T H \Delta z + \frac{1}{2} z_k^T H z_k + f^T z_k + f^T \Delta z \\ &= \frac{1}{2}\Delta z^T H \Delta z + (f + H' z_k)^T \Delta z + f(z_k) \\ &\leq \frac{1}{2} L \Delta z^T \Delta z + (f + H z_k)^T \Delta z + f(z_k) \end{aligned} \tag{3.18}$$

El algoritmo FISTA utiliza un punto de linealización distinto al anterior que se denota por $y_k = z_k - \Delta y$. En forma de pseudocódigo el algoritmo consta de los siguientes pasos:

1. $k \leftarrow 1, z_0 \leftarrow 0, y_1 \leftarrow 0, t_1 \leftarrow 1, FIN \leftarrow NO$
2. Repetir hasta $FIN = SI$
   a) Calcular el $\Delta y$ tal que

$$\begin{aligned} \min_{\Delta y} \quad & \frac{1}{2} L \Delta y^T \Delta y + (f + H y_k)^T \Delta y \\ s.a. \quad & |y_{ki} + \Delta y_i| \leq a_i, \quad i = 1, \dots, n \end{aligned}$$

   b) Hacer $z_k \leftarrow y_k + \Delta y$
   c) Hacer $t_{k+1} \leftarrow \frac{1}{2}\left(1 + \sqrt{1 + 4t_k^2}\right)$
   d) Hacer $y_{k+1} \leftarrow z_k + \frac{t_k - 1}{t_{k+1}}(z_k - z_{k-1})$
   e) Hacer $k \leftarrow k + 1$
   f) Si $|\Delta z_i| \leq TOL$, entonces $FIN \leftarrow SI$
3. $z^* \leftarrow z_k$

La cualidad de rápido (*Fast*) del algoritmo proviene de la mejora de uno previo (ISTA) utilizando un descenso de gradiente acelerado introducido por Yurii Nesterov (1983). En concreto el método corresponde a los pasos c) y d) del pseudocódigo. De una forma intuitiva se puede decir que el método realiza un paso de descenso de gradiente para ir de $y_k$ a $z_{k+1}$ para luego *deslizarse* un poco más allá de $z_{k+1}$ en la dirección dada por el anterior punto $z_k$.

---

**Ejemplo 3–5**. *Sea $z \in \mathbb{R}^n$ un vector de números reales, calcular la solución del problema*

$$\begin{aligned} \min_{z} \quad & \frac{1}{2} z^T H z + f^T z \\ s.a. \quad & |z_i| \in Z \\ & Az = b \end{aligned}$$

*Donde $Z = \{z \in \mathbb{R}^n : LB \leq z \leq UB\}$, $H$ una matriz diagonal tal que $H_{ii} > 0$, $A \in \mathbb{R}^{m \times n}$, con $m < n$ y*

$[A, b]$ *es de rango m (i.e. que las restricciones son linealmente independientes).*

Este es el ejemplo que realmente nos atañe, pues es lo que nos vamos a encontrar a la hora de minimizar la función objetivo del MPC.

Para resolver este problema se utiliza dualidad. Defínase

$$\begin{aligned} J(z) &= \frac{1}{2} z^T H z + f^T z \\ L(z,x) &= J(z) - x^T(Az - b) \\ z(x) &= \arg\min_{z \in Z} L(z,x) \\ f(x) &= L(z(x), x) \end{aligned} \tag{3.19}$$

La función $L(z,x)$ se denomina la función Lagrangiana del problema, $x$ es el multiplicador de Lagrange y $f(x)$ es la función dual.

Sea $z^*$ la solución óptima del problema original y $J^*$ el coste óptimo, entonces es claro que

$$J^* = L(z^*, x) \geq f(x) \tag{3.20}$$

Para todo $x$, pues $Az^* - b = 0$. Por lo tanto la función dual es una cota inferior del problema de optimización. El objetivo es pues obtener la mejor cota inferior, maximizando la función dual, i.e., resolviendo el problema

$$x^* = \arg\max_x f(x) \tag{3.21}$$

Definiendo $f^* = f(x^*)$, se tiene que $J^* \leq f^*$. El cálculo de $z(x)$ en este problema de optimización tiene una solución explícita directa, teniendo en cuenta que se puede escribir como

$$z(x) = \arg\min_{z \in Z} \frac{1}{2} z^T H z + (f - A^T x)^T z \tag{3.22}$$

Y que la matriz $H$ es diagonal y $Z$ es una caja. Por lo tanto para resolver el problema de optimización basta con resolver el problema dual para lo cual se utilizará el ya descrito algoritmo FISTA. Esto se basa en la siguiente desigualdad.

$$f(x + \Delta x) \geq f(x) - \Delta x^T (Az(x) - b) - \frac{1}{2} \Delta x^T W \Delta x \tag{3.23}$$

Siendo $W = AH^{-1}A^T$. Nótese que

$$\arg\max_{\Delta x} -\Delta x^T (Az(x) - b) - \frac{1}{2} \Delta x^T W \Delta x = \arg\min_{\Delta x} \Delta x^T (Az(x) - b) + \frac{1}{2} \Delta x^T W \Delta x \tag{3.24}$$

Dado que la matriz $W$ es invertible, por ser $H$ definida positiva y $A$ de rango completo por filas, el óptimo de este problema de optimización se alcanzará en el punto en el que se anula el gradiente, i.e., $(Az(x) - b) + W\Delta x = 0$, por lo que

$$\Delta x = -W^{-1}(Az(x) - b) \tag{3.25}$$

La implementación del algoritmo FISTA cambiaría levemente con respecto a la introducida anteriormente:

1. $k \leftarrow 1, x_0 \leftarrow 0, y_1 \leftarrow 0, t_1 \leftarrow 1, FIN \leftarrow NO, Q \leftarrow W^{-1}$
2. Repetir hasta $FIN = SI$
   a) Calcular $z(y_k)$ tal que

$$\min_{z \in Z} \quad \frac{1}{2} z^T H z + (f - A^T y_k)^T z$$

b) Si $|Az(y_k) - b| \leq TOL$, entonces $FIN \leftarrow SI$

c) Si no

i. Calcular $\Delta y \leftarrow -Q(Az(y_k) - b)$

ii. Hacer $x_k \leftarrow y_k + \Delta y$

iii. Hacer $t_{k+1} \leftarrow \frac{1}{2}\left(1 + \sqrt{1 + 4t_k^2}\right)$

iv. Hacer $y_{k+1} \leftarrow x_k + \frac{t_k - 1}{t_{k+1}}(x_k - x_{k-1})$

v. Hacer $k \leftarrow k + 1$

3. $z^* \leftarrow z_k$

La implementación en código MATLAB® de dicho algoritmo es la siguiente

**Código 3.2** Código MATLAB® : Algoritmo FISTA

```
x=zeros(1,(N+1)*n)';y=zeros(1,(N+1)*n)';t=1;FIN=0;x_ant=zeros(1,(N+1)*n)';
Qfis=inv(A/H*A');

kk=1;
while ~FIN

    z=saturacion(H,f-A'*y,lim);

    if norm(A*z-b)<=1e-6
        FIN=1;
    else
        incr_y=-Qfis*(E*z-b);
        x=y+incr_y;
        t_sig=0.5*(1+sqrt(1+4*(t^2)));
        y=x+(((t-1)/t_sig)*(x-x_ant));
        kk=kk+1;
        x_ant=x;
        t=t_sig;

    end
    if kk>2000
        FIN=1;
        disp('exceso de bucle')
    end
end
```

Es frecuente que para la búsqueda de ciertos valores óptimos el algoritmo FISTA necesite de numerosas iteraciones. Esto puede acarrearnos problemas en el control pues puede que no cumpla con los tiempos críticos. Para ello se añade al final del bucle una comprobación por la que, si se superan las 2000 iteraciones, el programa salga y devuelva la última $z^*$ calculada, no siendo el valor óptimo sino una aproximación.

### 3.1.2 Planteamiento de la función objetivo ajustada al problema de control

Como ya vimos en la sección anterior, la manera que tiene el bloque ***MPC*** para calcular la actuación $u$ es la minimización de la función objetivo (3.1). Pero dicha función es la representación formal, por lo que es necesario identificar los elementos del problema de control de forma que quede calculable con los métodos anteriores. Dicho de otro modo, lo que vamos a exponer a continuación es la adecuación de la formulación general del MPC, vista en (2.11), para su posible resolución mediante una minimización de una función

objetivo del tipo (3.1) y utilizando los mismos planteamientos que los ejemplos anteriormente resueltos.

Definiremos los siguientes elementos:

$L$: Coste de etapa

$$L(x,u) = x^T Q x + u^T R u \quad ; \quad Q,R > 0 \tag{3.26}$$

El modelo de predicción lineal:

$$x(i+1) = Ax(i) + Bu(i) \tag{3.27}$$

Las restricciones en forma de cajas:

$$\begin{aligned} X &= \{x: \quad LB_x \le x \le UB_x\} \\ U &= \{u: \quad LB_u \le u \le UB_x\} \end{aligned} \tag{3.28}$$

El planteamiento del problema es el siguiente:

$$\begin{aligned} \min_u \sum_{i=0}^{N-i} & x(i)^T Q x(i) + u(i)^T R u(i) \\ & x(0) = x \qquad (1) \\ & x(i+1) = Ax(i) + Bu(i) \qquad (2) \\ & LB_x \le x(i) \le UB_x \qquad (3) \\ & LB_u \le u(i) \le UB_u \\ & Ax(N-1) + Bu(N-1) = 0 \qquad (4) \end{aligned} \tag{3.29}$$

Y lo queremos de la siguiente forma:

$$\begin{aligned} \min_z \quad & z^T D z \\ s.a. \quad & LB_z \le z \le UB_z \\ & Ez = b(x) \end{aligned} \tag{3.30}$$

De manera que podamos resolverlo con los métodos usados para el **Ejemplo 3–5**. A continuación, iremos identificando elementos de (3.29) para que queden de la forma (3.30), para ello nos servirán los índices en color rojo.

En primer lugar definiremos el vector $z$ que estará formado por todo el conjunto de estados u actuaciones para un horizonte de predicción $N$ dado.

$$z = \begin{bmatrix} x(0) \\ u(0) \\ x(1) \\ u(1) \\ \vdots \\ x(N) \\ u(N) \end{bmatrix} \tag{3.31}$$

Donde $x(i)$ y $u(i)$ son

$$x(i) = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \qquad u(i) = \begin{bmatrix} u_1 \\ \vdots \\ u_m \end{bmatrix} \tag{3.32}$$

La matriz diagonal $D$ sería ($Q$ y $R$ son también diagonales)

$$D = \begin{bmatrix} Q & & & & \\ & R & & & \\ & & Q & & \\ & & & R & \\ & & & & \ddots \end{bmatrix} \tag{3.33}$$

Siendo $N(n+m) \times N(n+m)$ las dimensiones de la matriz. Los espacios en blanco representan elementos nulos.

Ya tenemos por lo tanto el primer elemento de (3.30), que correspondería al coste

$$V_N = \begin{bmatrix} x(0) \\ u(0) \\ x(1) \\ u(1) \\ \vdots \\ x(N) \\ u(N) \end{bmatrix}^T \begin{bmatrix} Q & & & & \\ & R & & & \\ & & Q & & \\ & & & R & \\ & & & & \ddots \end{bmatrix} \begin{bmatrix} x(0) \\ u(0) \\ x(1) \\ u(1) \\ \vdots \\ x(N) \\ u(N) \end{bmatrix} \tag{3.34}$$

A continuación mostraremos primero que forma tendría la restricción $Ez = b(x)$ para seguidamente describir las motivaciones de dicha forma.

$$\overbrace{\begin{bmatrix} I & & & & & & & \\ A & B & -I & & & & & \\ & & A & B & -I & & & \\ & & & & & \ddots & & \\ & & & & & & \ddots & \\ & & & & & A & B & \end{bmatrix}}^{E} \overbrace{\begin{bmatrix} x(0) \\ u(0) \\ x(1) \\ u(1) \\ \vdots \\ x(N-1) \\ u(N-1) \end{bmatrix}}^{z} = \overbrace{\begin{bmatrix} x \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}}^{b(x)} \tag{3.35}$$

(1): $x(0) = x$. Este elemento corresponde a la primera de las operaciones en la ecuación anterior:

$$\begin{bmatrix} 1 & & \\ & \ddots & \\ & & 1 \end{bmatrix} \begin{bmatrix} x_1(0) \\ \vdots \\ x_n(0) \end{bmatrix} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \rightarrow x(0) = x \tag{3.36}$$

(2): $x(i+1) = Ax(i) + Bu(i)$. Lo realizan las operaciones comprendidas entre la segunda y la penúltima fila de la matriz $E$. Comprobamos por ejemplo con la primera de las operaciones, i.e., $i = 0$

$$\begin{bmatrix} A & B & -I \end{bmatrix} \begin{bmatrix} x(0) \\ u(0) \\ x(1) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \rightarrow Ax(0) + Bu(0) = x(1) \tag{3.37}$$

(4): $Ax(N-1) + Bu(N-1) = 0$. Esta ecuación realmente denota que llegamos al origen, es la llamada restricción terminal. Corresponde a la última de las operaciones

$$\begin{bmatrix} A & B \end{bmatrix} \begin{bmatrix} x(N-1) \\ u(N-1) \end{bmatrix} = [0] \rightarrow Ax(N-1) + Bu(N-1) = 0$$

(3.38)

Restaría la restricción (3) que corresponde a las restricciones de la caja, operación que podemos representar con $LB_z \leq z \leq UB_z$ de la siguiente forma

$$\overbrace{\begin{bmatrix} LB_x \\ LB_u \\ LB_x \\ LB_u \\ \vdots \end{bmatrix}}^{LB_z} \leq \overbrace{\begin{bmatrix} x(0) \\ u(0) \\ x(1) \\ u(1) \\ \vdots \\ x(N) \\ u(N) \end{bmatrix}}^{z} \leq \overbrace{\begin{bmatrix} UB_x \\ UB_u \\ UB_x \\ UB_u \\ \vdots \end{bmatrix}}^{UB_z} \tag{3.39}$$

Por lo que ya estarían completamente identificados los elementos de (3.30) con la posibilidad de ser implementado con los algoritmos descritos en el **Ejemplo 3–5**. Faltaría construir los elementos del problema $D$, $z$, $LB_z$, $UB_z$, $E$ y $b(x)$. Para ello haremos uso de un código MATLAB® que presentaremos en capítulos posteriores cuando se conozcan todos los elementos del problema.

# 4 EL BUCLE MPC

*Aquel que duda y no investiga, se torna no sólo infeliz,*
*sino también injusto.*

*- Blaise Pascal -*

En el presente capítulo se describirá el bucle completo de control predictivo, con todos sus componentes nombrados en el **Resumen**. A modo de introducción, se ilustrará el bucle de manera esquemática en forma de diagrama de bloques. A lo largo del capítulo serán frecuentes las simulaciones sobre la planta de los cuatro tanques para probar la eficiencia de los distintos componentes.

## 4.1 Introducción

El lazo total de control se ilustra en la Figura 4-1.

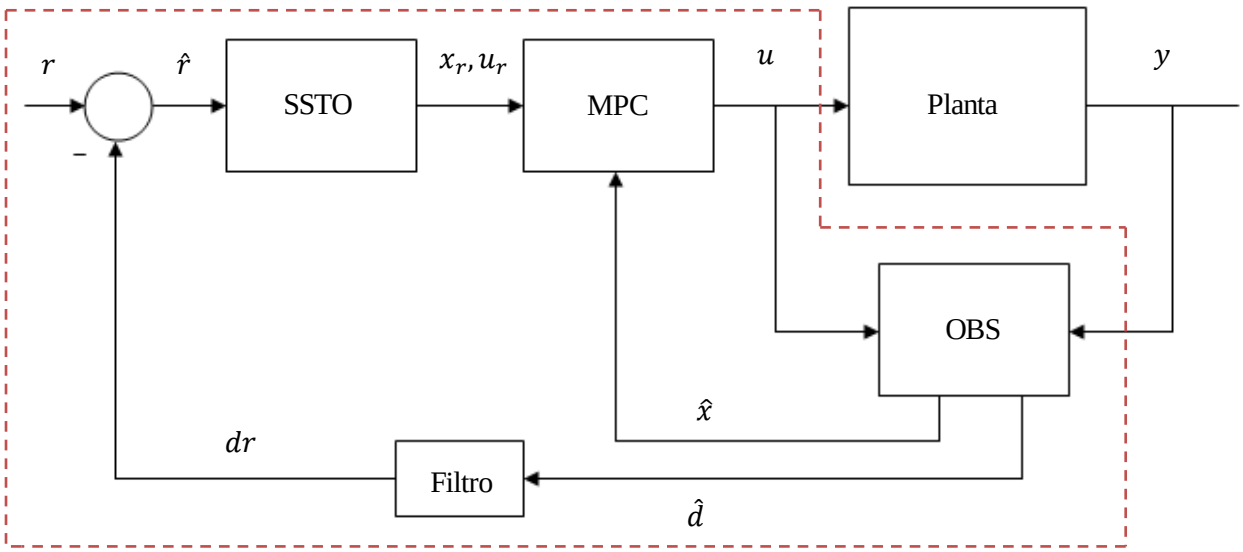



Figura 4-1. Bucle completo de control

En línea punteada roja se ha señalado el que sería el conjunto objetivo a implementar, quedando fuera el bloque que proporciona la salida, ya sea el modelo lineal de la planta, el simulador no lineal o la propia planta real.

De forma esquemática, se representa el bucle en la Figura 4-2 de manera que se diferencien claramente las entradas y salidas, y los parámetros con posibilidad de ajuste del MPC.

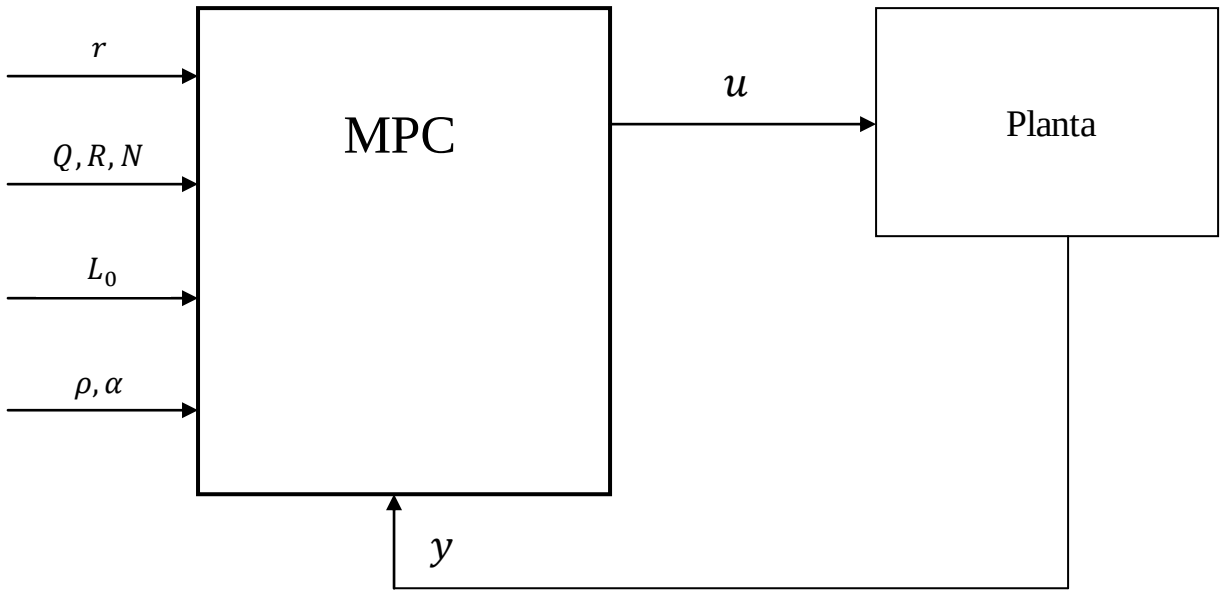



Figura 4-2. Diagrama esquemático del MPC

A la izquierda del bloque MPC se representan como entrada los parámetros ajustables de control, donde $r$ representa la referencia; $Q$, $R$ y $N$ son valores ya descritos en la **Sección 2.2**; $L_0$ es una variable ajustable del estimador de estado, el cual describiremos con detalle en secciones posteriores y $\rho$ y $\alpha$ corresponden al ajuste de los filtros. Cabe señalar en este punto que los únicos valor que pueden cambiarse **en línea**, i.e., que puede variarse su valor en plena actuación sin necesidad de cambio de parámetros significativos, son la referencia $r$ y los parámetros $\rho$ y $\alpha$.

El resto de valores, ya sean parámetros ajustables nombrados anteriormente o variables y valores internos del bucle de control, serán calculados previamente a la puesta en marcha del proceso de control. Esto quiere decir, que si queremos hacer un ajuste en algún parámetro diferente de la referencia o los filtros, tendremos que desacoplar el controlador y recalcular los valores, algo poco deseable debido a la dinámica tan lenta del sistema.

## 4.2 Componentes del bucle

Volviendo a la Figura 4-1 y a modo de enumeración de secciones posteriores describiremos brevemente la función de cada bloque:

- ***MPC***: Proporciona la actuación $u$ y recibe como entradas, $x_r, u_r$ del SSTO y $\hat{x}$ del estimador de estados. Su función es resolver el problema de optimización para la minimización de la función objetivo (2.13).
- ***OBS***: Es el estimador de estados. Se utiliza un observador de Luenberger que describiremos con más detalle en su correspondiente sección. Recibe como entradas la actuación $u$ de manos del bloque MPC y la salida $y$ de la planta. Proporciona el estado actual estimado $\hat{x}$ y la componente para ponderar perturbaciones y no linealidades $\hat{d}$.

- ***Filtros***[2]: Usando los parámetros $\rho$ y $\alpha$ se encargan de *suavizar* los cambios en la referencia y la variable $\hat{d}$ de forma que éstos no sean en forma de escalón sino de parábola, estrategia muy común en los controladores MPC.
- ***SSTO***: Recibe la referencia filtrada $\hat{r}$ y calcula el punto de equilibrio asociado a dicha referencia $x_r, u_r$. Su utilidad se verá con detalle más adelante, pero a modo de resumen se podría decir que su función es calcular puntos de equilibrio, o puntos óptimos para referencias no alcanzables.

### 4.2.1 Estimación de estados actuales

Como ya introdujimos en la **Sección 2.1.3**, en la mayoría de aplicaciones reales de control, el estado físico del sistema no puede ser determinado mediante observación directa. Para ilustrarlo mejor, consideremos el simple ejemplo de un vehículo por un túnel. Las velocidades y aceleraciones sólo son observables directamente a la entrada y salida del túnel, pero el valor exacto en todo momento dentro del túnel sólo puede ser estimado. Si un sistema es observable, es posible reconstruir totalmente el estado del sistema a través de sus medidas a la salida usando un estimador de estado.

Usaremos un planteamiento diferente al formulado en (2.10) al considerar

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k + w_k \\ y_k &= Cx_k \end{aligned} \tag{4.1}$$

Donde $w_k$ recoge todas las posibles perturbaciones y no linealidades del sistema y $y_k$ es simplemente la medida a la salida ($C$ es una matriz unidad). No es conocido el valor de $w_k$ por lo que haremos uso del Observador de Luenberger, el cual es un estimador en bucle cerrado que se define de la siguiente manera

$$\hat{x}_{k+1} = A\hat{x}_k + Bu_k + L_o(y_k - C\hat{x}_k) \tag{4.2}$$

Donde $\hat{x}$ representa el estado estimado, $u_k$ es conocido ya que lo calculamos previamente y $L_o$ es una matriz que ponderará la diferencia entre la salida medida $y_k$ y la salida que se estimaría $C\hat{x}_k$.

$$(y_k - C\hat{x}_k) \Longrightarrow C(x_k - \hat{x}_k) \Longrightarrow Ce_k \tag{4.3}$$

A continuación substraemos el valor del estado estimado (4.2) al modelado del estado real

$$\begin{aligned} & x_{k+1} = Ax_k + Bu_k + w_k \\ - \quad & \hat{x}_{k+1} = A\hat{x}_k + Bu_k + L_oCe_k \\ \hline & e_{k+1} = Ae_k - L_oCe_k + w_k \end{aligned} \tag{4.4}$$

Reordenamos el resultado

$$e_{k+1} = (A - L_oC)e_k + w_k \tag{4.5}$$

El observador de Luenberger para un sistema de tiempo discreto es por lo tanto asintóticamente estable cuando la matriz $(A - L_oC)$ tiene todos sus autovalores en el círculo unidad, i.e., elegimos $L_o$ de forma que se cumpla $L_o : |\lambda_i(A - L_oC)| < 1$.

Pero antes de abordar el cálculo de $L_o$ a continuación mostraremos como quedarían las ecuaciones del MPC:

[2] El filtro para la referencia no aparece representado en la Figura 4-1.

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k \\ y_k &= Cx_k + d_k \\ d_{k+1} &= d_k \end{aligned} \tag{4.6}$$

Y en forma matricial

$$\begin{aligned} \overbrace{\begin{bmatrix} x \\ d \end{bmatrix}_{k+1}}^{x_0^+} &= \overbrace{\begin{bmatrix} A & 0 \\ 0 & I \end{bmatrix}}^{A_0} \overbrace{\begin{bmatrix} x \\ d \end{bmatrix}_k}^{x_0} + \overbrace{\begin{bmatrix} B \\ 0 \end{bmatrix}}^{B_0} u_k \\ y_0 &= \overbrace{\begin{bmatrix} C & I \end{bmatrix}}^{C_0} x_0 \end{aligned} \tag{4.7}$$

Pero lo anterior se cumple para los estados reales de la planta que a priori son desconocidos. Debemos escribirlo para los estados estimados mediante el observador de Luenberger

$$\begin{bmatrix} \hat{x} \\ \hat{d} \end{bmatrix}_{k+1} = \overbrace{\begin{bmatrix} A & 0 \\ 0 & I \end{bmatrix}}^{A_0} \begin{bmatrix} \hat{x} \\ \hat{d} \end{bmatrix}_k + \overbrace{\begin{bmatrix} B \\ 0 \end{bmatrix}}^{B_0} u_k + \overbrace{\begin{bmatrix} L_x \\ L_d \end{bmatrix}}^{L_0} (y_k - C\hat{x}_k - \hat{d}_k) \tag{4.8}$$

Y volviendo a la notación que nos interesa

$$\begin{aligned} \hat{x}_{k+1} &= A\hat{x}_k + Bu_k + L_x(y_k - C\hat{x}_k - \hat{d}_k) \\ \hat{d}_{k+1} &= \hat{d}_k + L_d(y_k - C\hat{x}_k - \hat{d}_k) \end{aligned} \tag{4.9}$$

La cual ofrece la posibilidad de ser implementado fácilmente en código MATLAB®

**Código 4.1** Código MATLAB® : Observador de Luenberger

```
function [x_gorro,dr_gorro]=OBS(u,y,x_gorro,dr_gorro)
x_gorroant=x_gorro;
x_gorro=A*x_gorro+B*u+Lo(1:n,:)*(y-C*x_gorro-dr_gorro);
dr_gorro=dr_gorro+Lo(n+1:n+p,:)*(y-C*x_gorroant-dr_gorro);
end
```

Volviendo al cálculo de $L_0$ sabemos que, por propósitos del control, la salida del observador es realimentada al sistema a través de una ganancia $K$ : $u_k = -K\hat{x}_k$. Eligiremos dicha ganancia $K$ de forma que

$$\lambda(A_0 - L_o C_0) = \lambda(A_0^T - C_0^T L_0^T) = \lambda(A^* - C^* K) \tag{4.10}$$

Y que calcularemos mediante el diseño de un controlador cuadrático lineal para el sistema discreto

$$x_0^+ = A^* x_0 + B^* u_k \tag{4.11}$$

Para la $K$ que minimice:

$$\min \sum (x_i^T Q_0 x_i + u_i^T R_0 u_i) \tag{4.12}$$

Es importante señalar que $Q_0$ y $R_0$ no son las matrices de nuestro problema principal (3.1), sino otras que usaremos para ponderar la velocidad ($Q_0$) o la energía que estoy dispuesto a gastar ($R_0$). En nuestro caso haremos $R_0 = 0.1 \cdot Q_0$ y declarando $Q_0$ como la matriz identidad.

MATLAB® tiene funciones propias para resolver diseños de controladores cuadráticos lineales, por lo que usaremos el siguiente código para calcular $L_0 = K^T$

**Código 4.2** Código MATLAB® : Cálculo del $L_0$

```
Ao=[A,zeros(n,p);zeros(p,n),eye(p)]';
Co=[C,eye(p)]';
Qo=eye(n+p);
Ro=0.1*eye(m);
[Ko,null,EIG]=dlqr(Ao,Co,Qo,Ro);
Lo=Ko';
```

Tal como declaramos el problema en (4.6) y teniendo en cuenta el problema de *tracking* la ecuación final del sistema quedaría

$$x_r = Ax_r + Bu_r \qquad (4.13)$$
$$y_r = Cx_r + d_r = r$$

Y de forma matricial

$$\begin{bmatrix} A-I & B \\ C & 0 \end{bmatrix} \begin{bmatrix} x_r \\ u_r \end{bmatrix} = \begin{bmatrix} 0 \\ r-d_r \end{bmatrix} \qquad (4.14)$$

### 4.2.2 Obtención óptima de estados de equilibrio

Es la función que realiza el bloque ***SSTO***.

#### 4.2.2.1 Motivaciones teóricas

Como hemos definido anteriormente, $r(k) \in \mathbb{R}^p$ es la señal de referencia que debe seguir el sistema. El problema de seguimiento consiste en diseñar una ley de control $u(k) = \kappa(x(k),\mathbf{r})$ tal que:

- El bucle cerrado del sistema es estable en el sentido de Lyapunov:

**Teorema 4-1** (Teorema de Estabilidad de Lyapunov) *Un punto de equilibrio $\boldsymbol{x_0}$ de la ecuación diferencial homogénea $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x})$ es estable en el sentido de Lyapunov si todas las soluciones a la ecuación que parten en un entorno de $\boldsymbol{x_0}$ se mantienen cerca de $\boldsymbol{x_0}$ para todo tiempo posterior.*

- Las restricciones se cumplen para todo $k > 0$, $\big(x(k), u(k)\big) \in Z$.
- La salida converge a la referencia:

$$\lim_{k\to\infty} \|y(k) - r(k)\| = 0 \qquad (4.15)$$

El problema de seguimiento es intrínsecamente incierto, la futura evolución de la referencia debe suponerse constante en un valor.

Dada una referencia $r$, el punto de equilibrio de la planta $(x_r, u_r)$ debe calcularse satisfaciendo

- $x_r = f(x_r, u_r)$
- $(x_r, u_r) \in Z$
- $y_r = h(x_r, u_r)$ lo más cerca posible a $r$.

Y esto es lo que calculará el Optimizador de Estados de Equilibrio (SSTO) los cuales proporcionará al bloque ***MPC***.

La introducción de este problema de *tracking* ($y \to r$) hace necesario la inclusión de nuevos planteamientos. Dado el punto de equilibrio asociado a $r$, $(x_r, u_r)$ el problema ahora es

$$\begin{aligned} x^+ &= Ax_r + Bu_r \quad \rightarrow \quad \begin{bmatrix} A-I & B \\ C & 0 \end{bmatrix}\begin{bmatrix} x_r \\ u_r \end{bmatrix} = \begin{bmatrix} 0 \\ r \end{bmatrix} \\ r &= Cx_r \end{aligned} \tag{4.16}$$

Hace necesaria también la siguiente modificación:

$$\begin{aligned} x \rightarrow x_r \Longrightarrow \bar{x} = x - x_r = 0 \\ u \rightarrow u_r \Longrightarrow \bar{u} = u - u_r = 0 \end{aligned} \tag{4.17}$$

Algo que provoca la reescritura de la ecuación (3.29)

$$\begin{aligned} \min_u \sum_{i=0}^{N-i} \bar{x}^T Q\bar{x} + \bar{u}^T R\bar{u} \\ \bar{x}_0 = x - x_r \\ \bar{x}_{i+1} = A\bar{x}_i + B\bar{u}_i \\ \bar{x}_{min} \leq \bar{x} \leq \bar{x}_{max} \\ \bar{u}_{min} \leq \bar{u} \leq \bar{u}_{max} \\ \bar{x}_N = 0 \end{aligned} \tag{4.18}$$

Quedando el problema de programación a implementar:

$$\begin{aligned} \min_z \quad & z^T Dz + f(x_r, u_r)^T z \\ s.a. \quad & LB_z \leq z \leq UB_z \\ & Ez = b(x, x_r) \end{aligned} \tag{4.19}$$

Haciendo identificaciones similares a las realizadas para (3.30) habría que introducir el nuevo término $f(x_r, u_r)$

$$f = \begin{bmatrix} -Q \cdot x_r \\ -R \cdot u_r \\ -Q \cdot x_r \\ \vdots \end{bmatrix} \tag{4.20}$$

Ahora, al ser el último término $\bar{x}_N = 0 \rightarrow x_N = x_r$ , el vector $b(x, x_r)$ cambia en su último elemento

$$b = \begin{bmatrix} x \\ 0 \\ 0 \\ \vdots \\ x_r \end{bmatrix} \tag{4.21}$$

El resto de vectores $D$, $z$, $LB_z$ y $UB_z$ no varían.

#### 4.2.2.2 Cálculo de los puntos de equilibrio

Para dicho cálculo nos interesa que el origen esté dentro del conjunto de $X$ y $U$. Para ello simplemente restaremos el punto de linealización a las restricciones. Aseguramos así que el origen pertenezca al conjunto de estados posibles trabajando en variables incrementales.

Definimos ahora el conjunto $\Re$ formando por las referencias alcanzables. Es posible que existan referencias $r \notin \Re$ que no sea posible alcanzar, por lo que tendremos que aproximarnos a una referencia $\hat{r} \in \Re$ de manera que sea la referencia alcanzable más aproximada a la designada $r$.

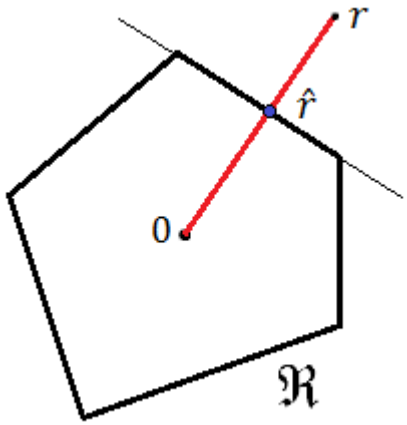



Figura 4-3. Referencia alcanzable $\hat{r}$ más próxima a la deseada

Lo plantearemos del siguiente modo: Al pertenecer el origen al conjunto de posibles referencias, definiremos un valor $\lambda \in (0,1)$ el cual modifique el valor de $r \to \hat{r} = \lambda r \to \hat{r} \in \Re$.

Pero antes haremos las siguientes declaraciones:

$$E = \begin{bmatrix} A - I & B \\ C & 0 \end{bmatrix} \qquad \begin{bmatrix} x_r \\ u_r \end{bmatrix} = M\theta + Lr \tag{4.22}$$

Y observando de nuevo (4.16) es fácil ver que

$$E(M\theta + Lr) = \begin{bmatrix} 0 \\ I \end{bmatrix} r \Longrightarrow \begin{cases} EM = 0 \\ EL = \begin{bmatrix} 0 \\ I \end{bmatrix} \end{cases} \tag{4.23}$$

A nosotros nos interesa el último término para hacer

$$L = E^{-1} \begin{bmatrix} 0 \\ I \end{bmatrix} \quad donde \quad \begin{bmatrix} x_r \\ u_r \end{bmatrix} = Lr \tag{4.24}$$

Donde el rango de $E$ es $n + m$ y debe cumplirse la condición de problema *cuadrado* $p = m$.

Ahora podemos definir el conjunto $\Re$ como

$$\Re = \left\{ r : \quad \begin{bmatrix} x_r \\ u_r \end{bmatrix} = Lr \quad ; \quad x_r \in X \, , u_r \in U \right\} \tag{4.25}$$

La matriz $L$ se puede tomar de la siguiente manera

$$L = \begin{bmatrix} L_x \\ L_u \end{bmatrix} \Longrightarrow x_r = L_x r \quad , \quad u_r = L_u r \tag{4.26}$$

Pudiéndose realizar la siguiente modificación en las restricciones de (4.25)

$$x_{min} \leq x_r \leq x_{max} \Longrightarrow x_{min} \leq L_x r \leq x_{max} \Longrightarrow \begin{cases} L_x r \leq x_{max} \\ -L_x r \leq -x_{min} \end{cases} \tag{4.27}$$

Y lo mismo para $u_r$ de forma que podemos redefinir (4.25) de la siguiente forma

$$\Re = \left\{ r: \quad \begin{bmatrix} L_x \\ -L_x \\ L_u \\ -L_u \end{bmatrix} r \leq \begin{bmatrix} x_{max} \\ -x_{min} \\ u_{max} \\ -u_{min} \end{bmatrix} \right\} = \left\{ r: \quad c_i^T r \leq d_i \quad \forall i \right\} \tag{4.28}$$

Aquí es donde entra en juego el valor $\lambda$ planteado al comienzo de la sección. Si se cumple $c_i^T r \leq d_i$ para todo $i$, la referencia estará dentro del conjunto $\Re$ de referencias alcanzables. Pero si para cualquier $i \in I_n$ dada ocurriera $c_i^T r > d_i$, donde $I_n$ es el conjunto de índices $i$ para los que ocurre lo anterior, habría que calcular un valor $\lambda$ para hacer $c_i^T \lambda r \leq d_i$ de forma que sirviera para todos los índices $i \in I_n$

$$c_i^T \lambda r \leq d_i \Longrightarrow \lambda \leq \frac{d_i}{c_i^T r} \Longrightarrow \lambda = \min_{i \in I_n} \left( \frac{d_i}{c_i^T r} \right) \tag{4.29}$$

Calculado de esta manera se garantiza que el valor $\lambda$ asegure la pertenencia de todas las $r$ al conjunto $\Re$. Los puntos de equilibrio se calcularían entonces

$$\begin{bmatrix} x_r \\ u_r \end{bmatrix} = Lr \cdot \lambda \tag{4.30}$$

En forma de pseudocódigo, el algoritmo a seguir por el SSTO sería:

1. Leer $r$
2. Comprobar restricciones $e_i = c_i^T r - d_i$
3. Si $e_i \leq 0 \quad \forall i \Longrightarrow \begin{bmatrix} x_r \\ u_r \end{bmatrix} = Lr$
4. Si no $I_n = \{ i: \ e_i > 0 \}$

   $$\lambda = \min_{i \in I_n} \left( \frac{d_i}{c_i^T r} \right) \Longrightarrow \begin{bmatrix} x_r \\ u_r \end{bmatrix} = Lr \cdot \lambda$$

Y el correspondiente código MATLAB®

**Código 4.3** Código MATLAB® : SSTO

```
function [xr,ur]=SSTO(r)
I=eye(n);
EE=[(A-I) B
     C    D];
L=EE\[zeros(n,m);eye(m)];
c=[L;-L];
d=[UBx;UBu;-LBx;-LBu];
e=c*r'-d;
indices=[];bandera=0;cont=0;lambda=1e6;
for i=1:(n+m)*2
    if e(i)>0
        indices=[indices i];
        bandera=1;
        cont=cont+1;
    end
end
if bandera==0
    REF=L*r';
    xr=REF(1:n,:);
    ur=REF(n+1:m+n,:);
else
    for i=1:cont
        aux=d(indices(i))/(c(indices(i),:)*r');
        if aux<lambda
```

```
            lambda=aux;
        end
    end
    REF=lambda*(L*r');
    xr=REF(1:n,:);
    ur=REF(n+1:m+n,:);
end
end
```

### 4.2.3 Filtros para el suavizado de $r$ y $\hat{d}$

En un MPC es conveniente que las transiciones en valores con posibilidad de cambios bruscos, tales como la referencia o la perturbación, deban ser de la manera más continua posible, i.e., evitando escalones convirtiéndolos en parábolas. Para ello haremos uso de un valor $\delta \in (0,1)$ de tal manera que, dado un valor cualquiera $g$, varíe de la siguiente forma

$$\hat{g}_{k+1} = (1-\delta)\hat{g}_k + \delta g \tag{4.31}$$

Así, si $g$ se incrementa su valor en un instante de muestreo, varíe de la siguiente forma

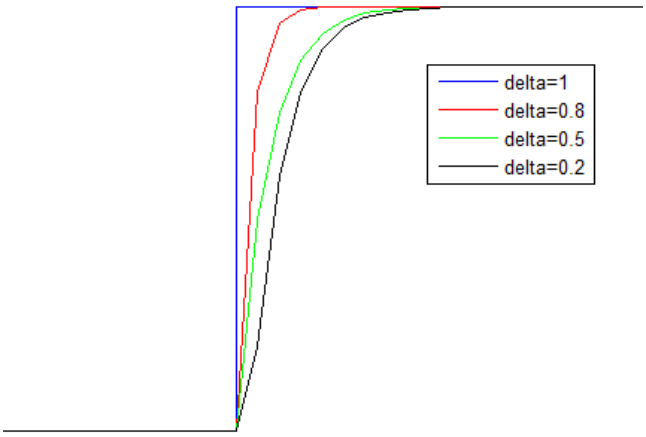



Figura 4-4. Paso de escalón a parábola mediante filtro

De esta manera filtraremos la referencia $r$ y las perturbaciones y no linealidades recogidas en $\hat{d}$ mediante los valores llamados $\rho$ y $\alpha$ respectivamente

$$\hat{r}_{k+1} = (1-\rho)\hat{r}_k + \rho r \tag{4.32}$$
$$dr_{k+1} = (1-\alpha)dr_k + \alpha\hat{d}$$

Quedando el bucle representado en la Figura 4-1 totalmente descrito.

# 5 SIMULACIONES

*El conocimiento es poder.*

*Francis Bacon*

Como dijimos en capítulos anteriores, nos apoyaremos en la potente herramienta MATLAB® para evaluar las diferentes estrategias anteriormente descritas antes de aplicar el MPC sobre un sistema real. Para ello, tomaremos los diferentes códigos expuestos y le daremos un carácter iterativo para simular el bucle de control con diferentes modificaciones. Validaremos dichas simulaciones con numerosas figuras y esquemas para su mejor exposición.

Aunque ya se ha comentado en la **Sección 2.3**, basaremos estas simulaciones principalmente sobre un modelo lineal y un simulador no lineal de la Planta de los 4 Tanques de los Laboratorios del Departamento de Automática.

## 5.1 Cálculo de las variables necesarias

A lo largo de los capítulos **3** y **4** presentamos numerosas ecuaciones y códigos que usaban valores, vectores y matrices formados por elementos del problema. Una gran mayoría de estos valores eran constantes a lo largo de dicho problema por lo que su cálculo sólo es necesario una vez y antes de poner en marcha el sistema.

Para ello haremos uso de un único código de MATLAB® que ejecutaremos previamente a cada simulación. Serán comunes los bucles `for` para el cálculo de las matrices. Se encargará tanto de calcular elementos constantes en el problema ($A$, $B$, $Q$, $R$, $E$…) como de inicializar elementos variables del mismo ($z$, $b$, $x_k$, $\hat{d}$, …).

Debido a su tamaño y a su trivialidad no lo expondremos aquí, aunque puede ser consultado en el **Código A.1**.

## 5.2 Ejecución y simulacion de códigos

Durante esta sección probaremos los diferentes elementos introducidos en sendos capítulos de una manera paralela al orden en los que fueron expuestos. Se ejecutarán incluyendo cada vez más elementos hasta completar el bucle completo de la Figura 4-1.

### 5.2.1 Ejecución del algoritmo FISTA

Como ya hemos dicho, MATLAB® posee innumerables funciones propias para realizar casi cualquier tipo de operación matemática. Entre ellas, posee `quadprog`, la cual puede resolver problemas de programación cuadrática, que es lo que a nosotros incumbe. Es aquí donde se hace necesario demostrar la eficiencia del algoritmo FISTA con respecto a la función de MATLAB® `quadprog`

Recordando el **Ejemplo 3-5**, a continuación mostraremos su solución con la función propia de MATLAB® y con el algoritmo detallado en dicho ejemplo. Para ello haremos $H = T^T T$ donde $T$ es una matriz aleatoria. Esto conseguirá que $H$ sea definida positiva. Las demás variables del problema las supondremos totalmente aleatorias respetando que se cumplan las dimensiones del problema

Ejecutamos:

```
tic
ejem5
toc
z
tic
quadprog(H,(f-(A'*y)),[],[],A,b,-a,a)
toc
```

Donde `tic` y `toc` sirven para mostrar los tiempos de ejecución de las funciones `ejem5` y `quadprog`.

El resultado por pantalla es el siguiente

```
Elapsed time is 0.000498 seconds.
z =
    0.0948
   -0.1099
   -0.0918
Optimization terminated.
ans =
    0.0948
   -0.1099
   -0.0918
Elapsed time is 0.008498 seconds
```

Podemos apreciar que, además de que el resultado es exactamente el mismo que con `quadprog`, el tiempo de ejecución es del orden del 5% del de `quadprog`, i.e., el algoritmo FISTA es hasta 20 veces más rápido que la función propia de MATLAB®, algo que nos resultará de gran utilidad a la hora de la ejecución real.

### 5.2.2 Ajuste de parámetros

En esta subsección mostraremos las consecuencias del ajuste de los diferentes parámetros introducidos en la Figura 4-2 sobre las respuestas de modelos de control.

#### 5.2.2.1 $N$, $Q$ y $R$

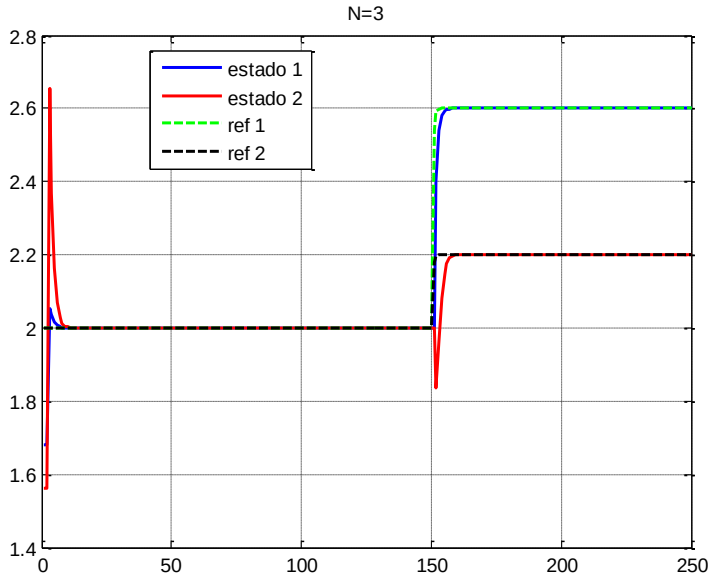
Para este apartado definiremos un sistema sencillo de $n = 3$ y $m = p = 2$. El modelo es el siguiente

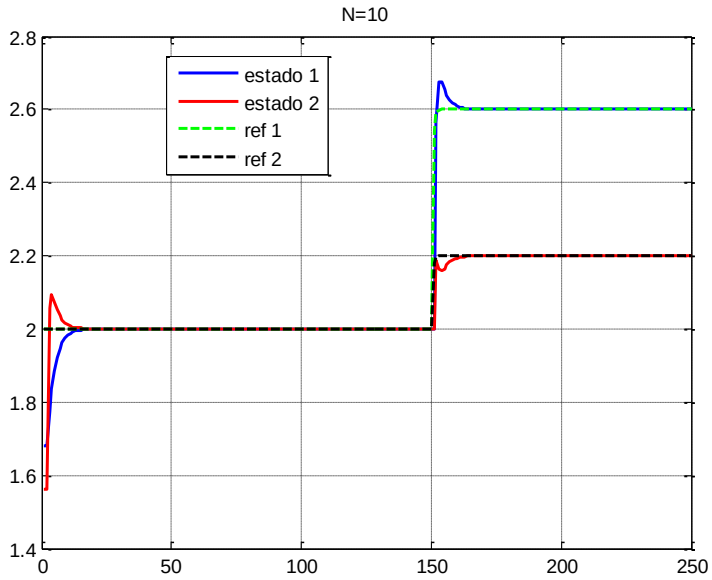
$$A = \begin{bmatrix} 0.8 & 0.1 & 0 \\ 0.1 & 0.7 & 0 \\ 0 & 0 & 0.85 \end{bmatrix} \qquad B = \begin{bmatrix} 0.5 & 0.1 \\ 1 & 0 \\ 0.1 & 0.7 \end{bmatrix} \tag{5.1}$$
$$|x_k| \leq 10$$
$$|u_k| \leq 10$$

Para su simulación usaremos un ***SSTO***, el cual simularemos y ajustaremos más adelante. Cabe señalar también que supondremos que los estados son directamente observables ($x^+ = Ax + Bu$) por lo que no dispondremos

de estimador de estado y serán representados directamente en la gráfica. Haremos uso también de los filtros anteriormente descritos $\alpha$ y $\rho$ que también serán justificados próximamente.

Se realizará un cambio de referencia de $[2\ 2]$ a $[2.6\ 2]$ y simularemos para diferentes cambios en el horizonte de predicción $N$, en concreto para $N = 3$, $N = 10$ y $N = 50$.




Y la salida por pantalla es la siguiente:

```
Elapsed time is 0.540776 seconds.
Elapsed time is 0.557683 seconds.
Elapsed time is 2.544873 seconds.
```

Para $N = 3$, $N = 10$ y $N = 50$ respectivamente ($N = 50$ representado en la siguiente página). De lo expuesto se puede deducir que para mayor horizonte de predicción $N$ mejores resultados, pero a la vez mayor tiempo de ejecución. Como veremos más adelante, realmente el horizonte de predicción $N$ solo afecta al coste de ejecución. Los posibles malos resultados que se aprecien son consecuencia del resto de valores ajustables que se describirán seguidamente. Al final de la sección haremos una mejor exposición de esto.

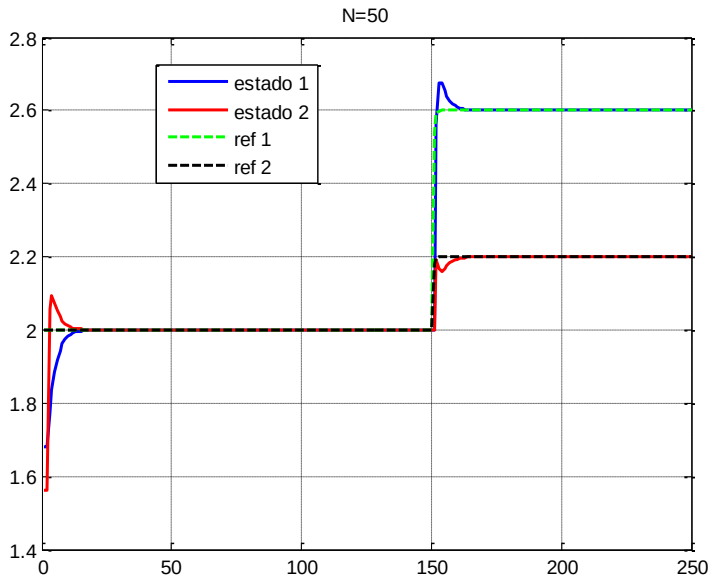
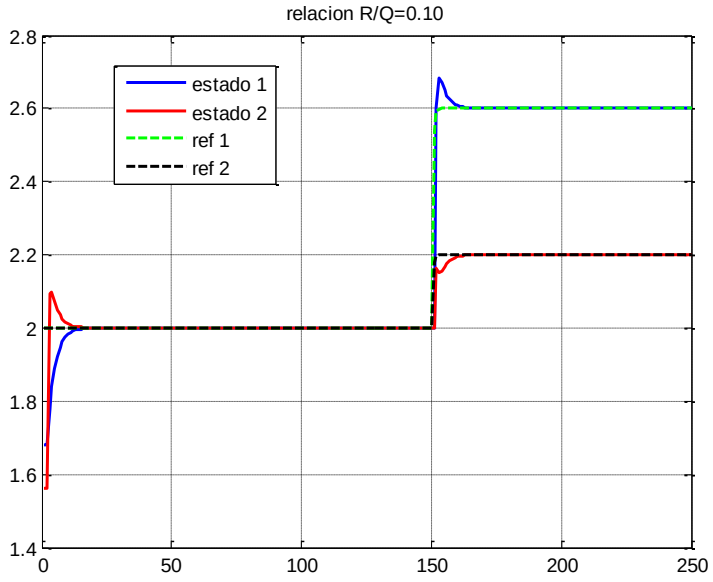



Figura 5-1. Respuestas para cambios en el horizonte de predicción $N$

A continuación mostraremos el ajuste sobre $Q$ y $R$. Aquí debemos decir que el problema se puede simplificar modificando simplemente la relación entre $R$ y $Q$.

Probaremos sobre las mismas condiciones anteriores para $N = 30$ y con unos cambios en la relación $R/Q$ de $R/Q = 0.1, R/Q = 1$ y $R/Q = 10$.




```
Elapsed time is 1.244494 seconds.
Elapsed time is 1.014895 seconds.
Elapsed time is 0.873580 seconds.
```

Es fácil observar que para $Q > R$ el sistema responde con mayores oscilaciones pero es más rápido, y al contrario, para $R > Q$ el sistema ofrece mejores respuestas oscilatorias pero alcanza más tarde la referencia. Esto es debido a, como ya dijimos en capítulos introductorios, que a mayor $Q$ más se esforzará el sistema en alcanzar la referencia de una manera rápida, resultando en un control muy agresivo, y para mayor $R$ más tendrá en cuenta las posibles limitaciones en la acción de control en detrimento del tiempo de llegada a $r$.

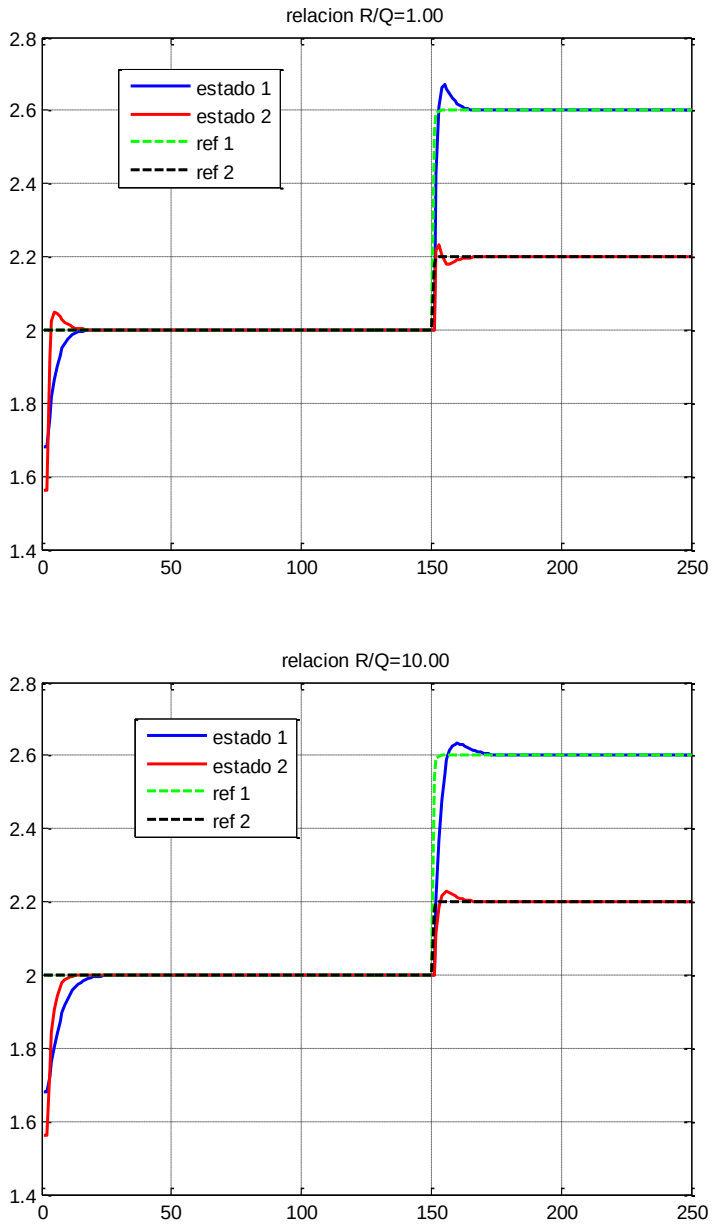



Figura 5-2. Respuestas para cambios en los parámetros de control $Q$ y $R$

Resultaría de vital interés poder cambiar estos parámetros en línea, ya que son los parámetros de control por excelencia. Pero esto traería consigo la necesidad de recalcular algunos valores bastante tediosos, entre ellos la matriz $Q$ perteneciente al algoritmo FISTA. Dada nuestra capacidad de cálculo bastante limitada (*Arduino*® Due) no nos será posible realizar este cambio en línea.

#### 5.2.2.2 Justificación del uso del *SSTO*

El algoritmo del ***SSTO*** no tiene realmente ningún parámetro ajustable, por lo que mostraremos su utilidad simplemente mostrando una simulación con y sin él.

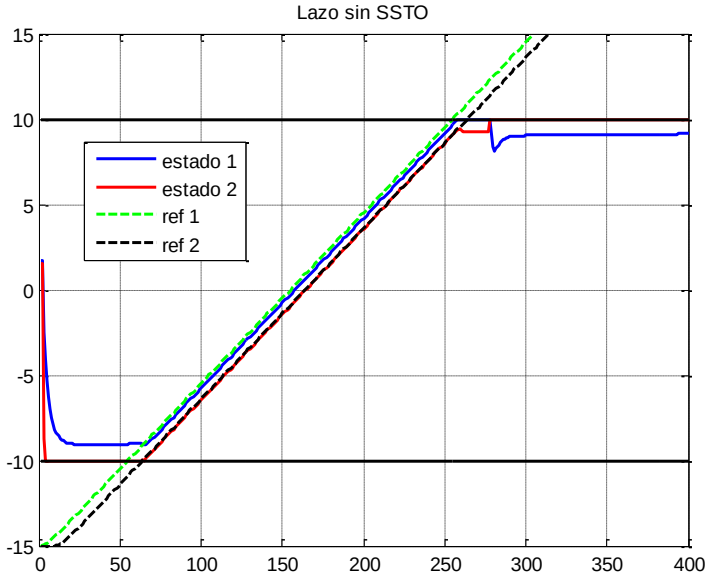
Como ya describimos en su correspondiente sección, el SSTO se encarga de proporcionar estados de equilibrio $[x_r, u_r]$ para posibles referencias no alcanzables. Por lo tanto, en un lazo de control de un problema de tracking en el que no exista SSTO, el cálculo del estado $[x_r, u_r]$ se realiza simplemente despejando la respectiva matriz del problema inicial de tracking:

$$\begin{bmatrix} A-I & B \\ C & 0 \end{bmatrix}\begin{bmatrix} x_r \\ u_r \end{bmatrix} = \begin{bmatrix} 0 \\ r \end{bmatrix} \rightarrow \begin{bmatrix} x_r \\ u_r \end{bmatrix} = \begin{bmatrix} A-I & B \\ C & 0 \end{bmatrix}^{-1}\begin{bmatrix} 0 \\ r \end{bmatrix} \tag{5.2}$$

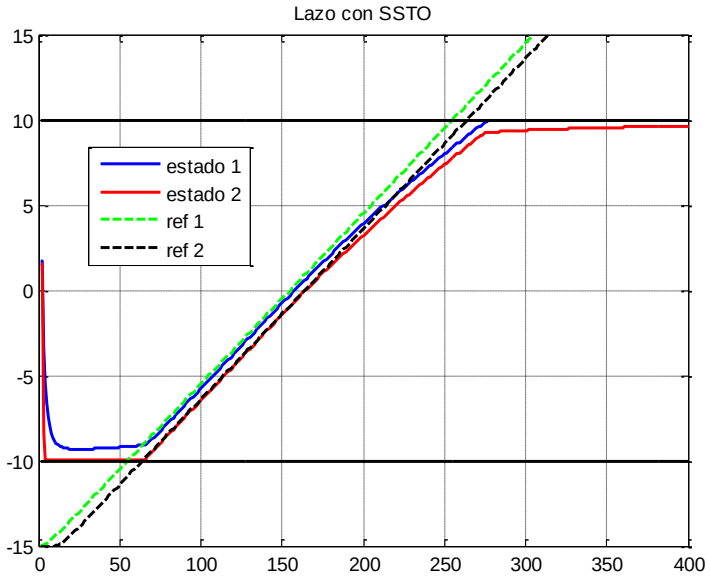
Que es fácilmente calculable en MATLAB®

```
REF=[A-eye(n) B;C D]\[zeros(n,1);r];
```

Las condiciones de simulación serán para $N = 20$ y $R/Q = 1$. Ahora, para ilustrar mejor la utilidad del SSTO, el cambio de referencia será sustancial. Comenzaremos con $[-15 - 15]$ y se irá incrementando cada tiempo de muestreo de manera que la referencia abarque tanto valores alcanzables como no alcanzables.




```
(...)
exceso de bucle
exceso de bucle
exceso de bucle
exceso de bucle
exceso de bucle
exceso de bucle
Elapsed time is 40.269335 seconds.
```



```
Elapsed time is 2.185715 seconds.
```

Figura 5-3. Respuesta con y sin SSTO

Como se puede apreciar en ambas figuras las respuestas son relativamente aceptables, pero como podemos ver en los tiempos de ejecución el sistema sin SSTO, al intentar calcular valores fuera de los límites posibles (límites de estado representados en negro en la figura anterior) satura la implementación del FISTA, que jamás cumple las tolerancias y sale por el exceso de 2000 bucles programado para estas situaciones. Esto incrementa considerablemente el tiempo de ejecución.

Esto es crítico en el control por lo que el bloque SSTO se convierte en vital en los problemas de control óptimo y predictivo.

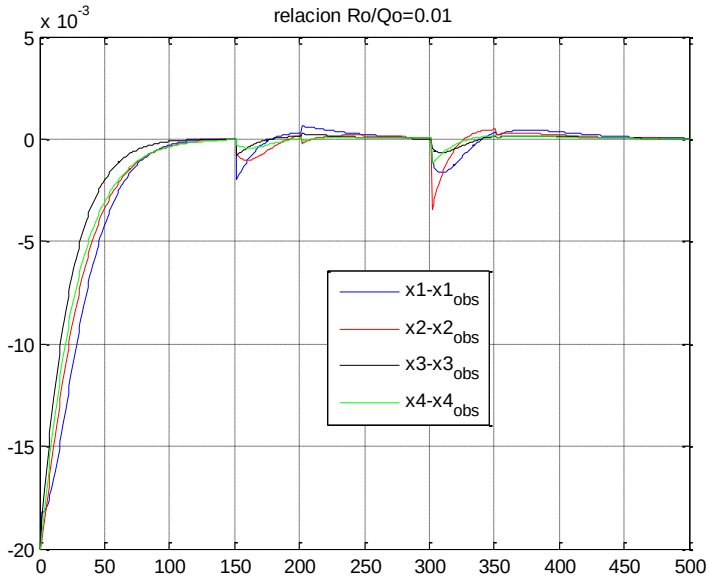
#### 5.2.2.3 El ajuste del estimador de estados

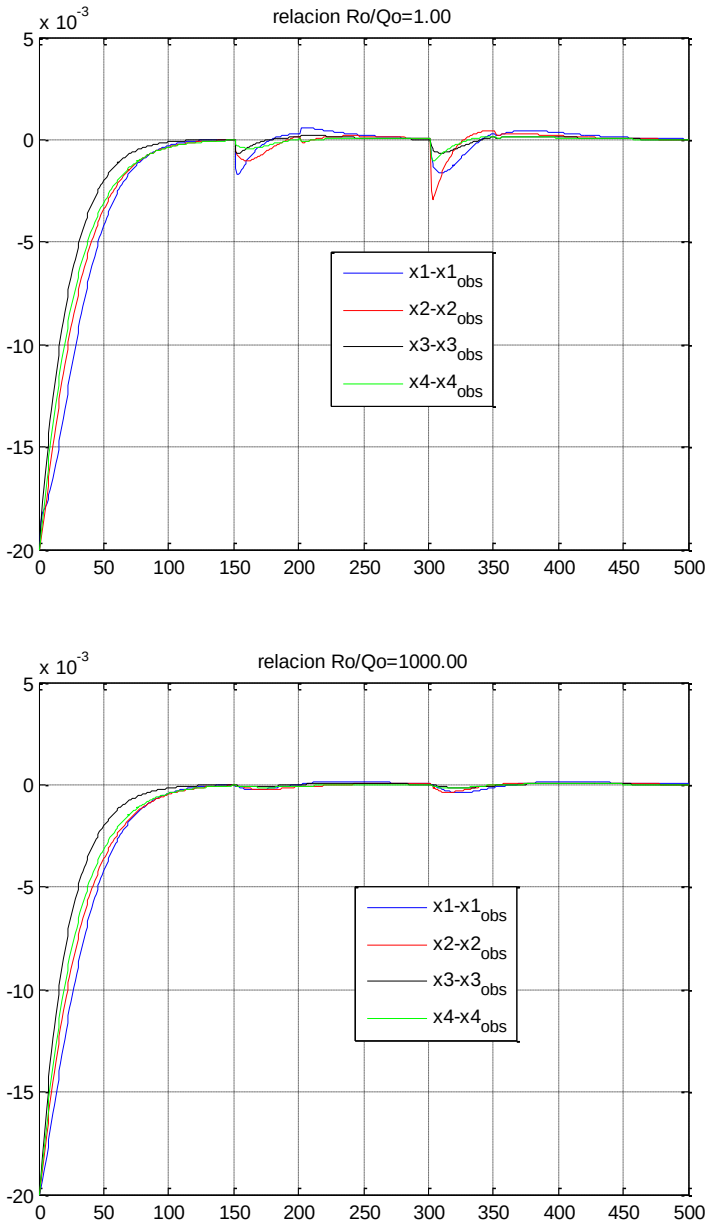
Como vimos en su correspondiente sección, el observador de Luenberger calcula mediante un problema de regulación cuadrática lineal un parámetro $L_o$ para la estimación de los estados. Dicho problema cuadrático utiliza de nuevo unas variables $Q_o$ y $R_o$ para su resolución, por lo que de nuevo convierte en verdaderamente ajustable la relación $R_o/Q_o$ e indirectamente $L_o$.

Para mostrar la bondad del estimador haremos uso en este caso y en los próximos del modelo lineal de la planta de los 4 tanques generado por **Código 2.1**. Es necesario suponer ruido en la lectura de la salida, por lo que introduciremos un valor `ruido` a la salida del sistema de dimensiones $p \times 1$ en forma de valores aleatorios entre -0.0015 y 0.0015 con cambios escalonados.

```
xk=A*xk+B*u;
yk=C*xk+ruido;
```

Inicialmente ni siquiera controlaremos, iremos introduciendo escalones en la actuación de la planta, calculando estados $\hat{x}$ estimados y representaremos la diferencia entre el estado real y el observado para diferentes valores de $R_o/Q_o$.

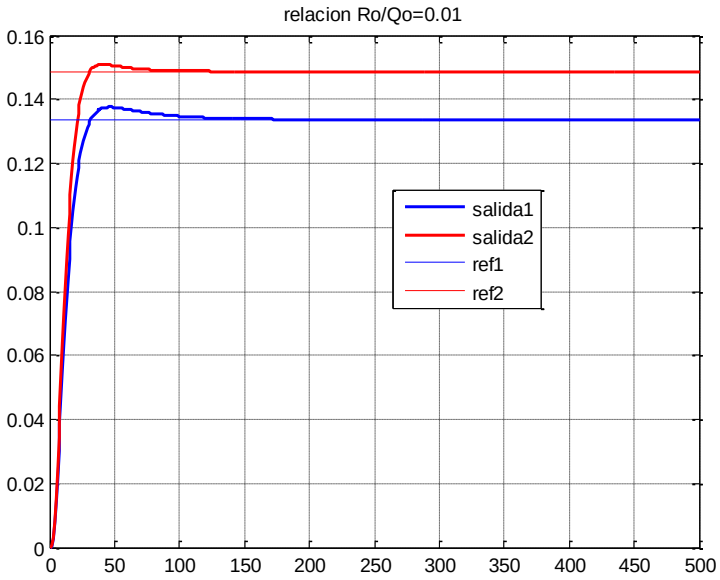


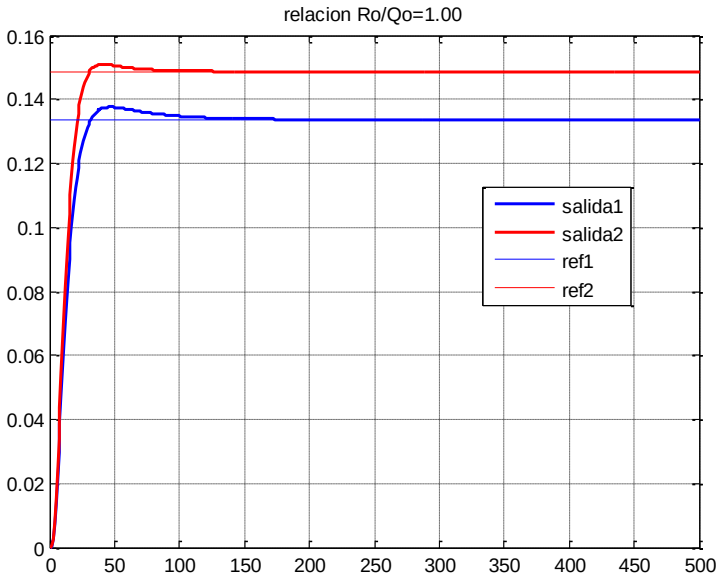



```
Elapsed time is 0.522774 seconds.
Elapsed time is 0.517753 seconds.
Elapsed time is 0.507712 seconds.
```

Figura 5-4. Estimación de estados para diferentes $L_o$

Podemos observar que, sin costes de ejecución asociados, para $R_o \gg Q_o$ la estimación del estado es óptima pero, como vamos a ver a continuación, en un MPC de alto horizonte de predicción estos valores apenas afectan al control. Ahora controlaremos el modelo de la planta de los 4 tanques con el MPC y aplicaremos los mismos ajustes anteriores para $N = 20$. Supondremos estado inicial 0 y fijaremos una referencia (en valores incrementales) de $[0.1337\ 0.1486]$.




Como podemos observar, el ajuste del estimador de estados apenas afecta al resultado del control ni al tiempo de ejecución. No obstante, es un elemento esencial en una aplicación real de MPC pues, además de que la mayoría de sistemas reales no son observables, un control MPC con bajo horizonte de predicción puede no actuar bien ante valores altos de $R_o$. Es por ello recomendable elegir siempre un estimador tal que $R_o > Q_o$ con la negativa de resultar en un control demasiado enérgico que provoque altas sobreoscilaciones y la saturación de la acción de control.

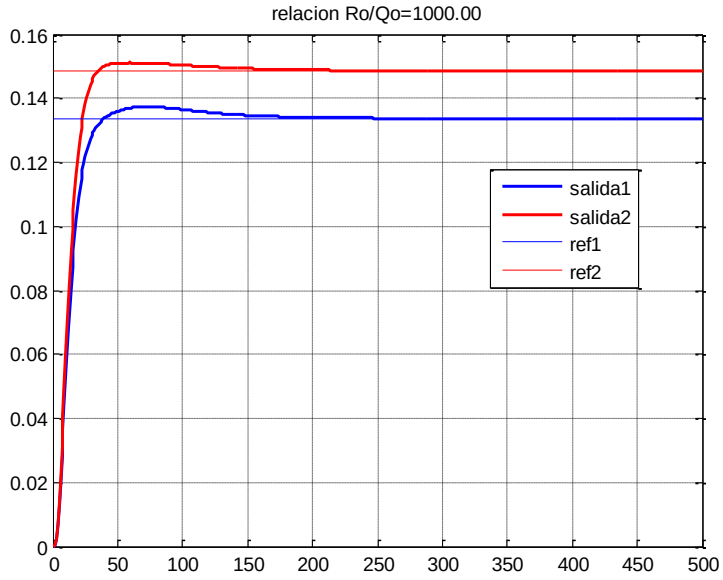
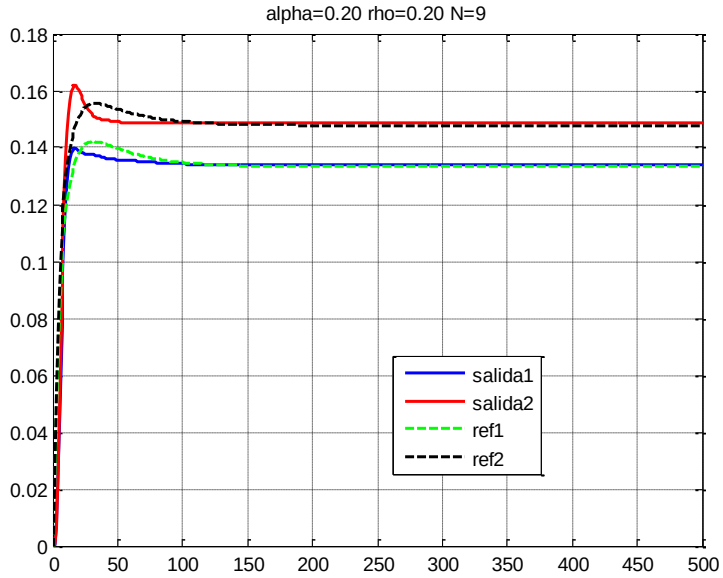



Figura 5-5. Ajuste del estimador en el control MPC de la planta

#### 5.2.2.4 Parámetros $\alpha$ y $\rho$ de los filtros

Es el último elemento que nos queda por ajustar del lazo antes de pasar al control no lineal. Es un elemento esencial, pues como ya dijimos en la **Sección 4.2.4**, los cambios escalonados en la referencia pueden afectar drásticamente a un MPC poco potente. Primero mostraremos resultados del control del modelo lineal, con el mismo ruido introducido anteriormente, para cambios en $\alpha$ y $\rho$ y para un horizonte de predicción de $\boldsymbol{N = 9}$.



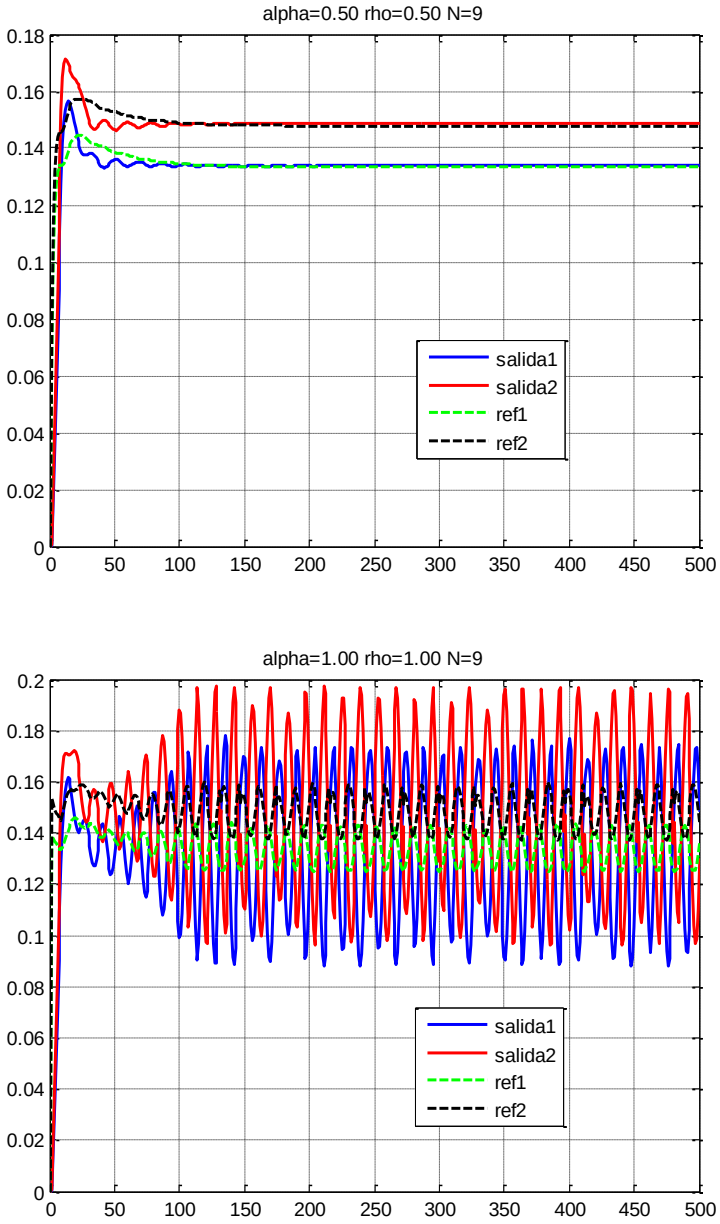



Figura 5-6. Ajuste de los filtros

Es claro ver que ante la ausencia de filtros, i.e., $\alpha, \rho = 1$, el sistema es totalmente incontrolable para un horizonte de predicción pequeño. De nuevo veremos que incrementando el horizonte de predicción el problema del control sin filtros se solventa. Así para $\boldsymbol{N = 30}$ y los filtros desactivados tenemos

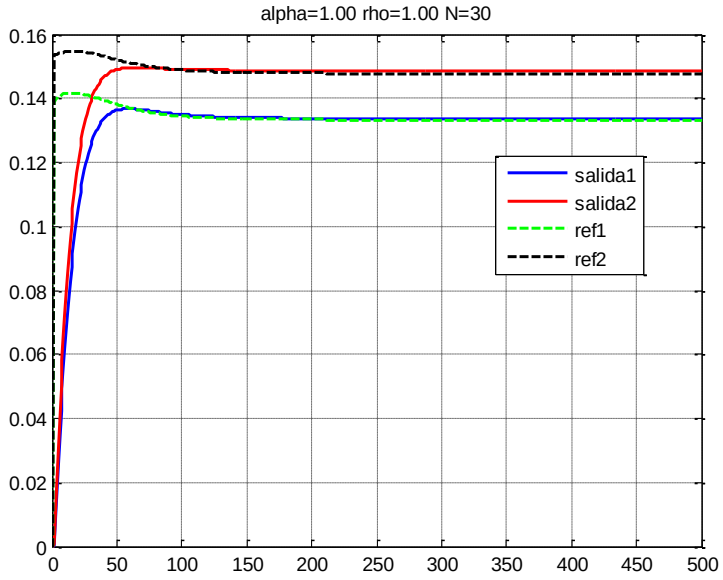



Figura 5-7. Control lineal sin filtros

Sin embargo, no es recomendable la ausencia de filtros en un control MPC, pues es probable que por cuestiones de computación de cálculo se deba bajar el horizonte de predicción.

### 5.2.3 La importancia del valor del horizonte de predicción $N$

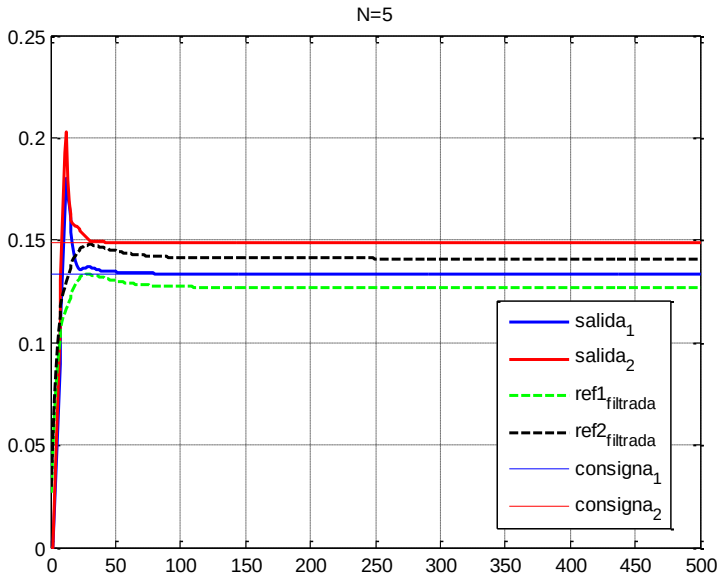
La conclusión de todas las subsecciones anteriores es que lo realmente **crítico** en un control MPC es el **horizonte de predicción $N$**. Un valor de $N$ alto mitigará las posibles consecuencias negativas que tengan tanto un observador de Luenberger muy agresivo como la ausencia de filtros (como hemos visto con anterioridad), además de estar en estrecha relación con las consecuencias de $Q$ y $R$. A cambio, el coste de ejecución se verá incrementado, algo a tener muy en cuenta en aplicaciones reales. Por el contrario si elegimos un valor de $N$ bajo debido posiblemente a nuestras limitaciones de cálculo y tiempo de ejecución, las consecuencias perjudiciales anteriormente descritas se verán incrementadas en severidad.

Es por tanto el verdadero reto del control predictivo, encontrar un balance entre el coste de ejecución y el resultado deseado, mediante una elección de $N$ adecuada a nuestras limitaciones y especificaciones de control.

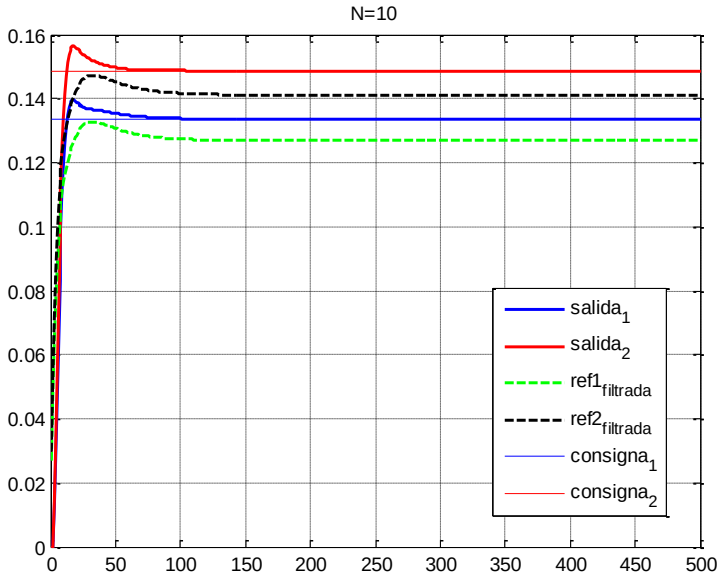
### 5.2.4 Implementación sobre el simulador no lineal

Como mostramos con el **Código 2.3**, tenemos a nuestra disposición un simulador no lineal de la planta de los cuatro tanques, que nos hará las veces de aplicación real al ser una buena aproximación de las condiciones reales con la introducción de la no linealidad.

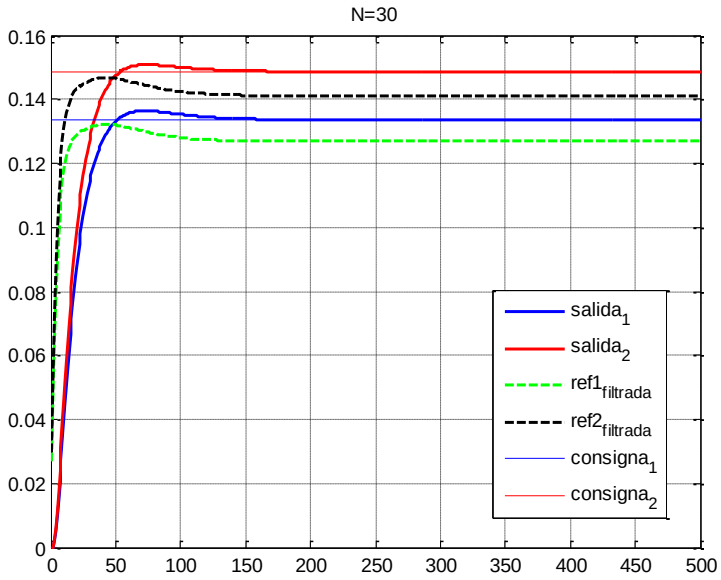
A continuación mostraremos simulaciones con el lazo completo para valores fijos de $\boldsymbol{R/Q = 0.1}$, $\boldsymbol{R_o/Q_o = 10}$, $\boldsymbol{\alpha = 0.2}$ y $\boldsymbol{\rho = 0.2}$. Variaremos el horizonte $\boldsymbol{N = 5, 10\ y\ 30}$




```
(...)
exceso de bucle
exceso de bucle
exceso de bucle
exceso de bucle
exceso de bucle
Elapsed time is 2.696580 seconds.
```




```
Elapsed time is 2.108455 seconds.
```




```
  Elapsed time is 3.407719 seconds.
```

Figura 5-8. Control del simulador no lineal para distintos $N$

Como ocurriera anteriormente con el modelo lineal, es necesario alcanzar un valor de compromiso para $N$, pues para $N = 5$ llega rápidamente a la referencia a costa de una acción de control muy saturada (sobreoscilaciones a la salida) y un coste de ejecución medio. Para $N = 30$ el control tiene un coste de ejecución alto, tarda en llegar a la referencia y ofrece pocas sobreoscilaciones.

#### 5.2.4.1 Problema: ceros de fase no mínima

Es sabido que el sistema de los cuatro tanques contiene ceros de fase no mínima [1] los cuales tienen una respuesta en frecuencia inversa. Esto quiere decir que cuando nosotros cambiemos la consigna de la referencia a otro valor, el sistema actuará en un primer momento de manera inversa al deseado para después corregir esta acción. Este problema se puede solventar no sin condiciones adversas en el tiempo de ejecución incrementando el horizonte de predicción $N$, aún así un cambio muy brusco en la referencia puede ocasionar oscilaciones inaceptables.

A continuación se simula para $\boldsymbol{N = 30}$ y con un cambio en la referencia de $r = [0.1337\ 0.1486] \rightarrow [0.14\ 0.138]$.

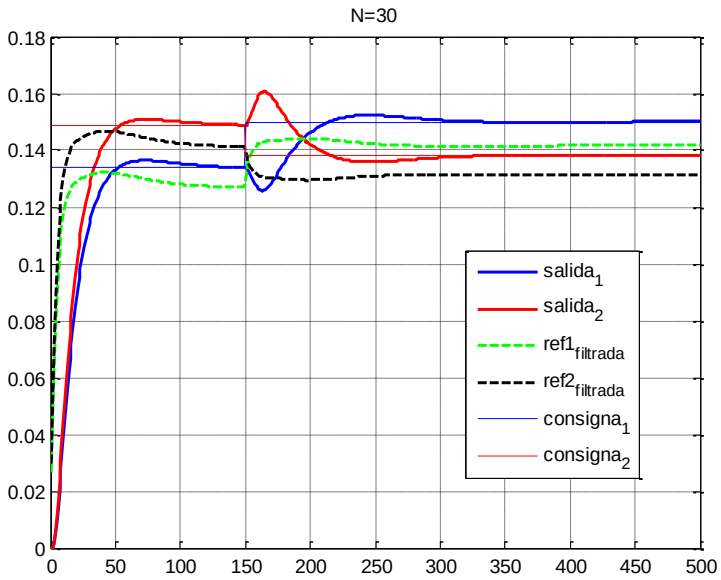



Figura 5-9. Cambio en la referencia sobre el modelo no lineal

Además de la respuesta inversa, podemos apreciar que, aún siendo el horizonte de predicción alto, las oscilaciones provocadas por el cambio de referencia son muy altas (~15%).

Una posible solución sería cambiar cada referencia por separado entre varios tiempos de muestreo. Así para el mismo cambio de referencia, pero esta vez cambiando primero la consigna de y1 y después, pasados unos tiempos de muestreo cambiar la consigna de y2.

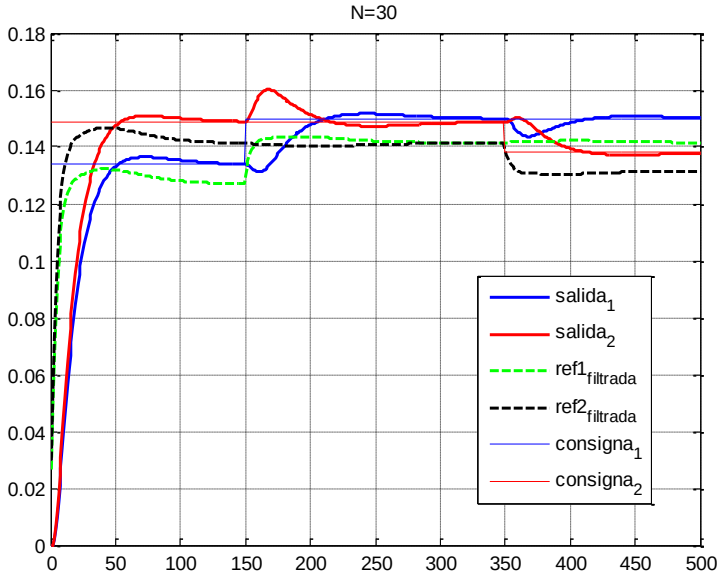



Figura 5-10. Cambio separado de referencias

Para el mismo cambio de consignas, la sobreoscilación relativa se reduce.

# 6 IMPLEMENTACIÓN EN *ARDUINO*®

Como dijimos al comienzo del documento, el objetivo final de este proyecto es la implementación del MPC en un microcontrolador tipo *Arduino*® Due, aunque gran parte del trabajo lo ocupe la simulación y prueba de resultados sobre la planta.

## 6.1 Introducción

El *Arduino*® Due es un microcontrolador basado en el procesador Atmel® SAM3X8E ARM Cortex-M3 [2] con un entorno de desarrollo integrado basado en el lenguaje de programación *Processing*, que soporta el lenguaje **C** y algunas estructuras de **C++**. La placa ofrece un conjunto de pines de entrada y salida, tanto analógicos como digitales, que pueden interactuar con otras placas o circuitos. El sistema proporciona también interfaces de comunicación serial, incluyendo comunicación USB, la cual permite interactuar con ordenadores personales.

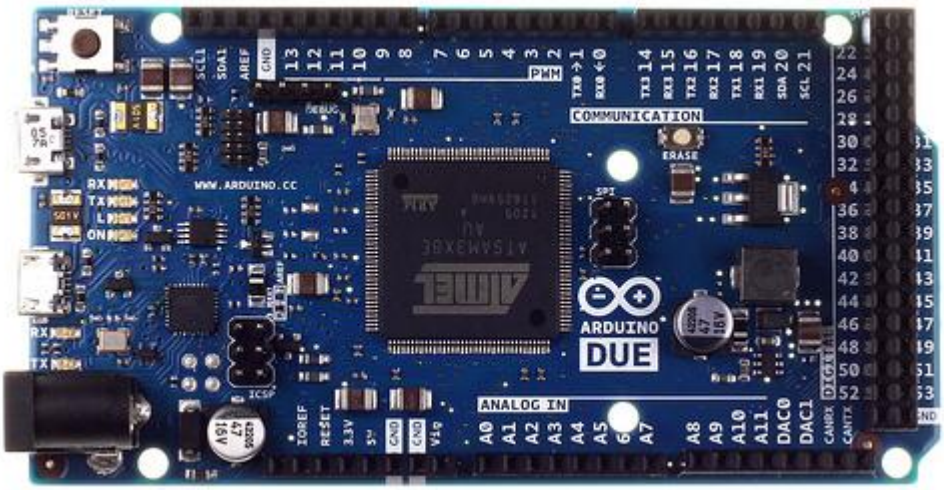



Figura 6-1. Placa *Arduino*® Due

Tabla 6-1. Características técnicas de la placa *Arduino*® Due

| Característica | |
|---|---|
| Voltaje de trabajo | 5 V |
| Voltaje de salida | 7 – 12 V |
| Velocidad CPU | 84 MHz |
| I/O Analógicas | 12/2 |
| I/O Digitales | 54 |
| PWM | 2 |
| SRAM | 96 kB |
| Flash | 512 kB |
| UART | 4 |

El módulo *Arduino*® está concebido como una herramienta de trabajo académico, es de licencia libre y utiliza hardware de código abierto, por lo que no es realmente usado en la industria. No obstante, para lo que nosotros ocupa en este proyecto nos será de utilidad para implementar en un entorno real lo que sería un sistema de Control Predictivo basado en Modelo, con potencial para adaptarse a cualquier sistema dado el modelo.

## 6.2 El problema del tiempo real en *Arduino*®

La computación en tiempo real se describe como sistemas sujetos a restricciones en *tiempo real*, por ejemplo, entre un evento y una respuesta del sistema. Se debe garantizar la respuesta dentro de unas restricciones de tiempo especificadas llamadas *deadlines*. Un sistema que no opera en tiempo real, generalmente no puede garantizar una respuesta dentro de un intervalo de tiempo.

Un sistema de tiempo real es descrito por Martin (1965) como aquel que "(…) controla un entorno recibiendo datos, procesándolos y enviando resultados mucho más rápido que el proceso bajo control, de forma que afecta al funcionamiento del entorno en ese intervalo de tiempo." [3]. El término *tiempo real* en otros ámbitos también puede significar que el reloj del sistema funciona a la misma velocidad que un reloj real o incluso significar "sin retraso significativo".

El software en tiempo real puede usar una o varias de estas características:

- Lenguaje de programación síncrona.
- Sistema operativo de tiempo real.
- Redes en tiempo real.

Las características anteriores proporcionan las estructuras esenciales para construir una aplicación software en tiempo real. El *Arduino*® Due carece de estas características o no son lo suficientemente eficientes como para considerarse un sistema en tiempo real. En secciones posteriores veremos cómo haremos frente a este problema mediante el uso de rutinas específicas de código.

## 6.3 El código C

Usaremos el lenguaje de programación **C** en el cual se basa el *Arduino*® entre otros. A lo largo de esta sección iremos comentando las principales características y diferencias de los códigos **C** que usemos con respecto al lenguaje propio de MATLAB®.

Debido a la similitud de lenguajes, no nos detendremos en la obtención de los valores dentro de los códigos, pues es análoga a la de MATLAB®. Describiremos simplemente las dificultades que se pueden encontrar al implementar en **C** y la simulación en un sistema que no es de tiempo real.

El objetivo final de la implementación es obtener una única función en **C** que reciba como entrada los datos

del problema, los parámetros de control, la salida $y$ del sistema a controlar y la referencia $r$, y devuelva la actuación $u$

```
void MPC(struct MPCData *Data, double r[p], double u[m], double yk[p])
```

Diseñaremos esa y otras funciones para cada bloque del sistema

```
void Calculo_U(struct MPCData *Data, double u[m]);
void PLANTA_NO_LINEAL(double u[m], struct MPCData *Data);
void observador(double u[m],double yk[p],struct MPCData *Data);
void SSTO(double r[p],struct MPCData *Data);
```

### 6.3.1 El coder de MATLAB®

Durante el trabajo hemos visto operaciones tediosas que debían hacerse con funciones propias de MATLAB® tales como la saturación en el problema de optimización o el SSTO.

Para su mejor implementación en **C** hemos hecho uso de una útil herramienta de MATLAB® llamada `coder` [5]. Dicha herramienta transforma códigos en lenguaje propio de MATLAB® a **C** o **C++**. No obstante en ocasiones el código estaba sobredimensionado o ni siquiera daba los resultados deseados, por lo que se tuvieron que hacer numerosas modificaciones.

Sin embargo el trabajo ahorrado es considerable por lo que es de merecida mención esta herramienta.

### 6.3.2 Cálculo previos de los valores invariables

Ya dijimos en la **Sección 4.1** que existen valores en el sistema que su cálculo solo es necesario al principio, antes de la ejecución del bucle. Entre ellos se encuentran valores que su cálculo conlleva un alto número de operaciones, tales como la matriz $E$ o la matriz $Q = W^{-1}$ del algoritmo FISTA, que puede tomar dimensiones de centenares de filas y columnas.

Como conocimos al comienzo de este capítulo el poder de cálculo del *Arduino*® es bastante limitado, pues no posee un procesador potente (aproximadamente 30 veces más lento que el de un ordenador convencional). Nos resultaría imposible en términos de coste de ejecución sólo el cálculo de la matriz $Q$ del algoritmo FISTA. Por ello haremos uso de la siguiente estrategia.

Siempre que se tenga que cambiar una variable que no se pueda variar en línea ($A$, $B$, $R/Q$, $N$…) se recalcularán todas con MATLAB® (**Código A.1**) y, en ese propio código, se generará un archivo `.txt` que inicialice una estructura con todos los valores que usará el código **C**. Las instrucciones serán del tipo

```
fprintf(datos,'#define N %d\n',N);
fprintf(datos,'#define n %d\n',n);
fprintf(datos,'#define p %d\n',p);
fprintf(datos,'#define m %d\n',m);
fprintf(datos,'struct MPCData{\n');
fprintf(datos,'\t\tdouble A[n*n];\n');
fprintf(datos,'\t\tdouble B[n*m];\n');
fprintf(datos,'\t\tdouble C[p*n];\n');
(...)
```

Lo que generará un código legible en **C** de la forma

```
#define N 20
#define n 4
#define p 2
#define m 2
struct MPCData{
      double A[n*n];
      double B[n*m];
      double C[p*n];
(...)
}Planta4T={{0.93732,0.00000,0.04662,0.00000,              /*A*/
            0.00000,0.93159,0.00000,0.03713,
```

```
            0.00000,0.00000,0.95338,0.00000,
            0.00000,0.00000,0.00000,0.96287},
(...)
```

El código completo puede consultarse en **Código A.5**

Como se aprecia, se hará uso de una estructura que contendrá todos los datos del problema y los parámetros de control a usar, que se pasará por referencia a las diferentes funciones del bucle

```
(...)
observador(u,yk,&Planta4T);
SSTO(r,&Planta4T);
Calculo_U(&Planta4T,u);
(...)
```

### 6.3.3 Solución al Tiempo Real

Introducido al comienzo del capítulo, este problema es de vital consideración pues la distinción entre los muestreos del MPC y los tiempos de integración del simulador no lineal (o tiempo de lectura de salidas para cualquier otra aplicación) es imprescindible para un control lógico y eficaz.

Una solución parcial es el llamado *Near Real-Time* o NRT (Cerca de Tiempo Real), que se utiliza en computación y control. Consiste en la introducción de retrasos entre la ocurrencia de un evento y el uso de los datos procesados.

En nuestro caso, los eventos diferenciables son: el cálculo de los estados cada tiempo de integración mediante el simulador no lineal (10ms) y el cálculo de la acción de control cada tiempo de muestreo (5000ms).

Sabemos que el *Arduino*® ejecuta sus instrucciones recogidas en la función propia `loop()` de forma iterativa y no concurrente. Sus ciclos no pueden conocerse pero si se puede conocer el tiempo actual. A partir del tiempo actual dividiremos la ejecución en ciclos para cada evento. Para su mejor comprensión se representa a continuación en forma de pseudocódigo:

```
Funcion loop() //funcion propia de Arduino
{
        Leo r
        Simulo x1,x2,x3,x4
        Leo salidas
        MPC
}
Simulacion x1,x2,x3,x4()
{
        static k=0
        Leo t1=tiempo actual
        SI (t1 > k*10ms)
        {
                Calculo x1,x2,x3,x4
        }
        k++
}
MPC()
{
        static k=1
        static dif=0
        Leo t1=tiempo actual
        SI ( (t1+dif) > k*5000ms)
        {
                Calculo u
                Leo t2=tiempo actual
                dif=t2-t1
        }
        k++
}
```

De esta manera el cálculo de los estados del sistema se realiza cada 10ms y el de la actuación cada 5000ms. Esto no son tiempos reales.

Por ejemplo después de las primeras 500 integraciones (comienzan en 0ms y terminan en 5000ms) se ejecuta la función MPC. El tedioso cálculo de la actuación le lleva a *Arduino*® 1500ms, que lo almacena en `dif`, por lo que el MPC saldrá en el instante 6500ms. Es aquí cuando entra la importancia del valor `dif` pues, si no se hiciera uso de él, el MPC volvería a entrar en el instante 10000ms. Recordemos que salió en el 6500ms, por lo que el simulador solo podrá haber calculado 350 integraciones y no las 500 para las que está concebido el control. Para poder calcular dicho número antes de que entre el MPC, el tiempo de ejecución debería llegar hasta los 11500ms, valor que se consigue sumando `dif` a los ciclos del MPC.

### 6.3.4 Cambios en línea de la referencia

Resulta vital en cualquier tipo de control la posibilidad de cambiar la consigna de las salidas mientras el sistema a controlar está funcionando. En cualquier otro caso no se trataría realmente de control.

En *Arduino*® nos encontramos con el problema de sus tiempos de ejecución como hemos visto en el apartado anterior. Necesitamos por lo tanto implementar la posibilidad de un cambio en las referencias, en el cual haya tiempo suficiente para poder pensar y escribir el valor sin que esto afecte para nada a la ejecución del programa.

Para ello definiremos una variable `cambio` la cual pueda dársele un valor de 1 o 2 en función de que se desee cambiar la referencia en el estado 1 o 2. En un primer momento, el operario pulsará el botón de la referencia a cambiar y el código seguirá su rutina sin modificación sustancial en el tiempo de ejecución. Desde ese momento el programa consultará un código introducido por una condición en el que se consultará si hay algún valor en el *buffer* del serial de la placa. Dicho código tampoco consume tiempo sustancial. Cuando el operario haya introducido el valor y se haya almacenado en el *buffer* el anterior programa simplemente actualizará la referencia y volverá a poner la variable `cambio` a cero.

La implementación de lo anterior puede consultarse en el **Código B.1**.

## 6.4 Simulación no lineal con *Arduino*®

Probaremos con los siguientes parámetros: $\boldsymbol{N = 20 \quad R/Q = 3/5 \quad R_o/Q_o = 10 \quad \alpha = 0.2 \quad \rho = 0.2}$

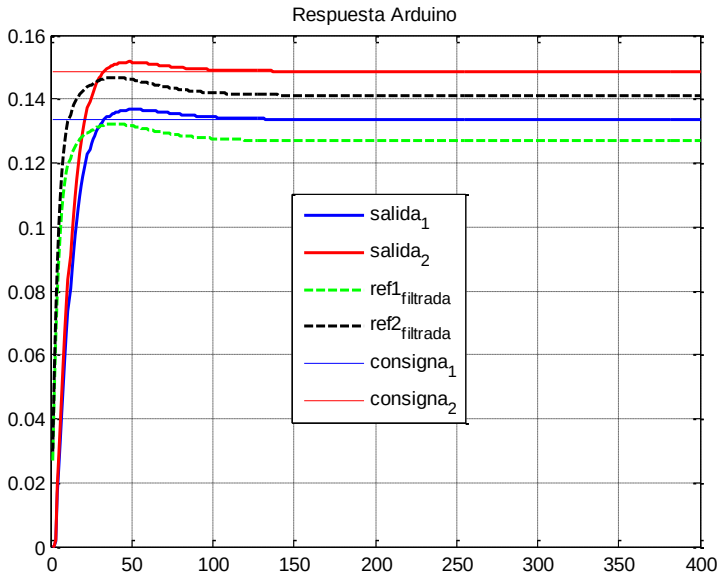



Figura 6-2. Respuesta control no lineal *Arduino*®

Para la representación se ha hecho uso de la comunicación serial entre MATLAB® y *Arduino*® (UART).

**Código 6.1** Código MATLAB® : Comunicación serial MATLAB® - *Arduino*®

```
clear
arduino=serial('COM9','BaudRate',9600);

fopen(arduino);
result=zeros(1,6);
bucles=400;

for i=1:bucles*6
    result=[result fscanf(arduino,'%f')];
end
fclose(arduino);
dimx6=length(result);
y1=result(7:6:dimx6);
r1f=result(8:6:dimx6);
c1=result(9:6:dimx6);
y2=result(10:6:dimx6);
r2f=result(11:6:dimx6);
c2=result(12:6:dimx6);

x=1:bucles;

plot(x,y1,'b',x,y2,'r',x,r1f,'--g',x,r2f,'--k','LineWidth',2);
hold on
grid on
plot(x,c1,'b',x,c2,'r')
```

Y la parte correspondiente en *Arduino*®

```
  (...)
    for(i=0;i<m;i++){
            Serial.println(yk[i],5);
            Serial.println(Data->r_rampa[i],5);
            Serial.println(r[i],5);
                    }
  (...)
```

El problema de la representación con MATLAB® es que dificulta la posibilidad del cambio de la referencia en línea. No se implementará la posibilidad de hacerlo en este trabajo puesto que para una aplicación real con control mediante *Arduino*® no se usaría MATLAB® para la representación puesto que ya existiría la posibilidad apreciarlo en el sistema real.

No obstante dos posibles soluciones serían; el envío de los cambios de consigna a través de MATLAB® mediante el comando `fpintf` y su correspondiente lectura en *Arduino*® con `Serial.Read`, o la construcción de un circuito botonera conectado a los pins analógicos y digitales que posee el *Arduino*®.

En cualquier caso, los cambios en la referencia serán posibles en nuestro sistema mediante el serial propio de *Arduino*® de la manera descrita en la sección anterior.

# CONCLUSIONES Y LÍNEAS FUTURAS

## Conclusiones

Tras todo lo desarrollado anteriormente se obtiene como producto final de trabajo un controlador predictivo basado en modelo (MPC), de amplio uso en la industria de procesos, implementado en un microcontrolador con la posibilidad de adecuarse al control de cualquier sistema dados los datos del problema $(A, B, C, D, x_o, u_o)$[3] y sus restricciones.

Al trabajarse en el lenguaje de programación **C**, el código diseñado podrá exportarse a casi la totalidad de procesadores del mercado trabajen o no en tiempo real. Las aplicaciones por lo tanto son numerosas.

## Líneas futuras

Se quedan en el tintero cuestiones como el cambio en línea de las relaciones $R/Q$ y $R_o/Q_o$ y por supuesto de $N$ lo que lo convertiría en un control completamente ajustable en ejecución. Para ello precisaríamos de una potente capacidad de cálculo, algo que no teníamos con *Arduino*® Due.

La aplicación en la planta real de los cuatro tanques se planteó desde un principio pero el devenir de acontecimientos ajenos y la acumulación de problemas en el propio diseño del lazo lo hizo imposible.

Queda por tanto pendiente la implementación de un control predictivo en un procesador que nos ofrezca mayores posibilidades y la consiguiente aplicación sobre un sistema(s) real(es).

[3] Solo para problemas cuadrados $p = m$. Se podrán controlar tantos estados como actuadores tenga el sistema.

# Índice de Códigos

Apéndice A

# Códigos MATLAB®

**Código A.1** Código MATLAB® : Generación de las variables necesarias

```
global A B C D n m p Lo LBx UBx LBu UBu Q R b xk z N E H rui
N=30; %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
q = [1.7602; 1.8072];
Uf=q;
Xf = puntoFunc(q);
[A, B, C, D] = modelo4tanques( Xf, 5);
r = [0.1337 0.1486];
LBx=0.2-Xf; % limite superior e inferior de X
UBx=1.2-Xf;
LBu=-q; % limite superior e inferior de U
UBu=3-q;
[fil,n]=size(A);
[filb,m]=size(B);p=m;
Q=eye(n);
R=relacionRQ*eye(m);
xk=zeros(n,1);
lim=zeros((n+m)*N,1);
for i=1:n+m:(n+m)*N
    lim(i:i+n-1)=UBx;
end
for i=n+1:n+m:(n+m)*N
    lim(i:i+m-1)=UBu;
end

%% OBSERVADOR

x_gorro=zeros(n,1);
dr_gorro=0;

Ao=[A,zeros(n,p);zeros(p,n),eye(p)]';
% Bo=[B;zeros(p,m)];
Co=[C,eye(p)]';
Qo=eye(n+p);
Ro=relacionRoQo*eye(m);
[Ko,null,EIG]=dlqr(Ao,Co,Qo,Ro);
Lo=Ko';
t_est=5/log(sum(EIG));
```

```
%% FILTROS

dr=0;alpha=0.2;
r_rampa=0;rho=0.2;

%% Variables para MPC

H=zeros(N*(n+m));
E=zeros((N+1)*n,(n+m)*N);
I=eye(n);cont=0;
for i=1:2*N
    if flag==1
       for aux=1:m
           H(i+cont+aux-1,i+cont+aux-1)=R(aux,aux);flag=0;
       end
       cont=cont+m-1;
    else
       for aux=1:n
           H(i+cont+aux-1,i+cont+aux-1)=Q(aux,aux);flag=1;
       end
       cont=cont+n-1;
    end
end

contIn=0;contIm=0;
for i=1:N
    if i==1
        E(1:n,1:n)=I;
    else
        E(1+contIn:n+contIn,1+contIn+contIm:n+contIn+contIm)=-I;
    end
    contIn=contIn+n;
    contIm=contIm+m;
end
contAn=0;contAm=0;
for i=1:N
    E(n+1+contAn:2*n+contAn,1+contAn+contAm:n+contAn+contAm)=A;
    contAn=contAn+n;
    contAm=contAm+m;
end
contBn=0;contBm=0;
for i=1:N
    E(n+1+contBn:2*n+contBn,n+1+contBn+contBm:n+m+contBn+contBm)=B;
    contBn=contBn+n;
    contBm=contBm+m;
end

b=zeros((N+1)*n,1);

ruido=[(0.03*rand-0.015)*ones(1,200),(0.03*rand-
0.015)*ones(1,150),(0.03*rand-0.015)*ones(1,150)...
    ;(0.03*rand-0.015)*ones(1,200),(0.03*rand-0.015)*ones(1,150),(0.03*rand-
0.015)*ones(1,150)];
```

**Código A.2** Código MATLAB® : Modelo lineal

```
function yk=PLANTA(u)
global A B C xk rui
xk=A*xk+B*u;
yk=C*xk+rui;
end
```

**Código A.3** Código MATLAB® : Actualización de los valores del MPC

```
        Qxr=Q*xr;
        Rur=R*ur;
        nn=0;mm=0;
        for i=1:N
            f(1+nn+mm:n+nn+mm,1)=-Qxr;
            f(n+1+nn+mm:n+m+nn+mm,1)=-Rur;
            nn=nn+n;mm=mm+m;
        end
        b((N+1)*n-n+1:(N+1)*n)=xr;
        b(1:n,1)=x_gorro;
        calc
        u=z(n+1:n+m,1);
```

**Código A.4** Código MATLAB® : Bucle para el control

```
datosPlanta;histy=[xk(1:p,:)];histx_gorro=[x_gorro ];u=[0;0];histu=[u ];
histdr=[ ];histref=[ ];
for k=1:bucles
        xknl=simula_NL_4Tanques(xk+Xf,u+Uf,5,0.01)';
        xk=xknl-Xf;
        yk=C*xk;
        [x_gorro,dr_gorro]=OBS(u,yk,x_gorro,dr_gorro);
        dr=(1-alpha)*dr+alpha*dr_gorro;
        r_gorro=r-dr';
        r_rampa=(1-rho)*r_rampa+rho*r_gorro;
        [xr,ur]=SSTO(r_rampa);
        MPC;

        histy=[histy yk];histx_gorro=[histx_gorro x_gorro];
        histdr=[histdr dr];histu=[histu u];histref=[histref r_rampa'];
end

%% REPRESENTACION POR PANTALLA
```

Hay que señalar que el **Código A.4** es genérico, i.e., sirve para la simulación de cualquier lazo de control. El que se expone corresponde al lazo completo con el simulador no lineal. Si se deseara simular una configuración diferente se modificarían los elementos del bucle necesarios.

**Código A.5** Código MATLAB® : Generación de la estructura en .txt

```
datos=fopen('datos.txt','w+');

%%
%% SE DEBE HABER EJECUTADO CON ANTERIORIDAD A.1
%%

AB_I=zeros(n,2*n+m);
for i=1:n
    for j=1:n
        AB_I(i,j)=A(i,j);
    end
end
for i=1:n
    for j=1+n:n+m
        AB_I(i,j)=B(i,j-n);
    end
end
for i=1:n
    for j=n+m+1:n+n+m
        AB_I(i,j)=-I(i,j-n-m);
    end
end


At=A';Bt=B';
ET1=zeros(n+m,2*n);
for i=1:n
    for j=1:n
        ET1(i,j)=I(i,j);
    end
end
for i=1:n
    for j=1+n:n+n
        ET1(i,j)=At(i,j-n);
    end
end
for i=n+1:n+m
    for j=n+1:n+n
        ET1(i,j)=Bt(i-n,j-n);
    end
end

At=A';Bt=B';
ET2=zeros(n+m,2*n);
for i=1:n
    for j=1:n
        ET2(i,j)=-I(i,j);
    end
end
for i=1:n
    for j=1+n:n+n
        ET2(i,j)=At(i,j-n);
    end
end
for i=n+1:n+m
    for j=n+1:n+n
        ET2(i,j)=Bt(i-n,j-n);
    end
end
```

```
Qfis=inv(E/H*E');
b=zeros((N+1)*n,1);



%% ---------------DATOS SSTO------------------------
I=eye(n);
EE=[(A-I) B
     C   D];
L=EE\[zeros(n,m);eye(m)];
c=[L;-L];
d=[UBx;UBu;-LBx;-LBu];


%% ---------------CARGA DATOS C----------------------
fprintf(datos,'#define N %d\n',N);
fprintf(datos,'#define n %d\n',n);
fprintf(datos,'#define p %d\n',p);
fprintf(datos,'#define m %d\n',m);
fprintf(datos,'struct MPCData{\n');
fprintf(datos,'\t\tdouble A[n*n];\n');
fprintf(datos,'\t\tdouble B[n*m];\n');
fprintf(datos,'\t\tdouble C[p*n];\n');
fprintf(datos,'\t\tdouble D[p*m];\n');
fprintf(datos,'\t\tdouble LBx[n*1];\n');
fprintf(datos,'\t\tdouble UBx[n*1];\n');
fprintf(datos,'\t\tdouble LBu[m*1];\n');
fprintf(datos,'\t\tdouble UBu[m*1];\n');
fprintf(datos,'\t\tdouble Q[n*1];\n');
fprintf(datos,'\t\tdouble R[m*1];\n');
fprintf(datos,'\t\tdouble lim[(n+m)*1];\n');
fprintf(datos,'\t\tdouble H[(n+m)*1];\n');
fprintf(datos,'\t\tdouble AB_I[n*(2*n+m)];\n');
fprintf(datos,'\t\tdouble ET1[(n+m)*2*n];\n');
fprintf(datos,'\t\tdouble ET2[(n+m)*2*n];\n');
fprintf(datos,'\t\tdouble Qfis[((N+1)*n)*((N+1)*n)];\n');
fprintf(datos,'\t\tdouble Lo[(n+p)*p];\n');
fprintf(datos,'\t\tdouble L[(n+m)*p];\n');
fprintf(datos,'\t\tdouble c[(2*(n+m))*p];\n');
fprintf(datos,'\t\tdouble d[(2*(n+m))*1];\n');
fprintf(datos,'\t\tdouble x_gorro[n];\n');
fprintf(datos,'\t\tdouble xk[n];\n');
fprintf(datos,'\t\tdouble dr_gorro[p];\n');
fprintf(datos,'\t\tdouble dr[p];\n');
fprintf(datos,'\t\tdouble Xf[n];\n');
fprintf(datos,'\t\tdouble Uf[m];\n');
fprintf(datos,'\t\tdouble Tm;\n');
fprintf(datos,'\t\tdouble h;\n');
fprintf(datos,'\t\tdouble r_rampa[p];\n');
fprintf(datos,'\t\tdouble alpha;\n');
fprintf(datos,'\t\tdouble ro;\n');
fprintf(datos,'\t\tdouble xr[n];\n');
fprintf(datos,'\t\tdouble ur[m];\n');
fprintf(datos,'\t\tdouble zmpc[N*(n+m)];\n');



fprintf(datos,'}Planta4T={{');%A
for i=1:n
    if i~=1 fprintf(datos,'\t\t\t');end
    for j=1:n
        if i==n && j==n
```

```
            fprintf(datos,'%.5f},\n',A(i,j));
        elseif i==1 && j==n
            fprintf(datos,'%.5f,\t\t/*A*/\n',A(i,j));
        elseif j==n
            fprintf(datos,'%.5f,\n',A(i,j));
        else
            fprintf(datos,'%.5f,',A(i,j));
        end
    end
end
fprintf(datos,'\t\t\t{');%B
for i=1:n
    if i~=1 fprintf(datos,'\t\t    ');end
    for j=1:m
        if i==n && j==m
            fprintf(datos,'%.5f},\n',B(i,j));
        elseif i==1 && j==m
            fprintf(datos,'%.5f,\t\t/*B*/\n',B(i,j));
        elseif j==m
            fprintf(datos,'%.5f,\n',B(i,j));
        else
            fprintf(datos,'%.5f,',B(i,j));
        end
    end
end
fprintf(datos,'\t\t\t{');%C
for i=1:p
    if i~=1 fprintf(datos,'\t\t    ');end
    for j=1:n
        if i==p && j==n
            fprintf(datos,'%.1f},\n',C(i,j));
        elseif i==1 && j==n
            fprintf(datos,'%.1f,\t\t/*C*/\n',C(i,j));
        elseif j==n
            fprintf(datos,'%.1f,\n',C(i,j));
        else
            fprintf(datos,'%.1f,',C(i,j));
        end
    end
end
fprintf(datos,'\t\t\t{');%D
for i=1:p
    if i~=1 fprintf(datos,'\t\t    ');end
    for j=1:m
        if i==p && j==m
            fprintf(datos,'%.1f},\n',D(i,j));
        elseif i==1 && j==m
            fprintf(datos,'%.1f,\t\t/*D*/\n',D(i,j));
        elseif j==m
            fprintf(datos,'%.1f,\n',D(i,j));
        else
            fprintf(datos,'%.1f,',D(i,j));
        end
    end
end
fprintf(datos,'\t\t\t{');%LBx
for i=1:n
    if i==1 fprintf(datos,'%.5f,\t\t/*LBx*/\n',LBx(i));
        elseif i==n fprintf(datos,'\t\t    %.5f},\n',LBx(i));
        else fprintf(datos,'\t\t    %.5f,\n',LBx(i));
    end
end
fprintf(datos,'\t\t\t{');%UBx
```

```
for i=1:n
    if i==1 fprintf(datos,'%.5f,\t\t/*UBx*/\n',UBx(i));
        elseif i==n fprintf(datos,'\t\t    %.5f},\n',UBx(i));
        else fprintf(datos,'\t\t    %.5f,\n',UBx(i));
    end
end
fprintf(datos,'\t\t\t{');%LBu
for i=1:m
    if i==1 fprintf(datos,'%.5f,\t\t/*LBu*/\n',LBu(i));
        elseif i==m fprintf(datos,'\t\t    %.5f},\n',LBu(i));
        else fprintf(datos,'\t\t    %.5f,\n',LBu(i));
    end
end
fprintf(datos,'\t\t\t{');%UBu
for i=1:m
    if i==1 fprintf(datos,'%.5f,\t\t/*UBu*/\n',UBu(i));
        elseif i==m fprintf(datos,'\t\t    %.5f},\n',UBu(i));
        else fprintf(datos,'\t\t    %.5f,\n',UBu(i));
    end
end
fprintf(datos,'\t\t\t{');%Q
for i=1:n
    if i==1 fprintf(datos,'%.1f,\t\t/*Q*/\n',Q(i,i));
        elseif i==n fprintf(datos,'\t\t    %.1f},\n',Q(i,i));
        else fprintf(datos,'\t\t    %.1f,\n',Q(i,i));
    end
end
fprintf(datos,'\t\t\t{');%R
for i=1:m
    if i==1 fprintf(datos,'%.1f,\t\t/*R*/\n',R(i,i));
        elseif i==m fprintf(datos,'\t\t    %.1f},\n',R(i,i));
        else fprintf(datos,'\t\t    %.1f,\n',R(i,i));
    end
end

fprintf(datos,'\t\t\t{');%lim
for i=1:n+m
    if i==1 fprintf(datos,'%.5f,\t\t/*lim*/\n',lim(i));
        elseif i==n+m fprintf(datos,'\t\t    %.5f},\n',lim(i));
        else fprintf(datos,'\t\t    %.5f,\n',lim(i));
    end
end

fprintf(datos,'\t\t\t{');%H
for i=1:n
    if i==1 fprintf(datos,'%.5f,\t\t/*H*/\n',H(i,i));
    else fprintf(datos,'\t\t    %.5f,\n',H(i,i));
    end
end
for j=n+1:m+n
    if j==m+n fprintf(datos,'\t\t    %.5f},\n',H(j,j));
    else fprintf(datos,'\t\t    %.5f,\n',H(j,j));
    end
end

fprintf(datos,'\t\t\t{');%[A B -I]
for i=1:n
    if i~=1 fprintf(datos,'\t\t    ');end
    for j=1:2*n+m
        if i==n && j==2*n+m
            fprintf(datos,'%.5f},\n',AB_I(i,j));
```

```
        elseif j==2*n+m && i==1
            fprintf(datos,'%.5f,\t\t/*[A B -I]*/\n',AB_I(i,j));
        elseif j==2*n+m
            fprintf(datos,'%.5f,\n',AB_I(i,j));
        else
            fprintf(datos,'%.5f,',AB_I(i,j));
        end
    end
end

fprintf(datos,'\t\t\t{');%ET1
for i=1:n+m
    if i~=1 fprintf(datos,'\t\t    ');end
    for j=1:2*n
        if i==n+m && j==2*n
            fprintf(datos,'%.5f},\n',ET1(i,j));
        elseif j==2*n && i==1
            fprintf(datos,'%.5f,\t\t/*ET1*/\n',ET1(i,j));
        elseif j==2*n
            fprintf(datos,'%.5f,\n',ET1(i,j));
        else
            fprintf(datos,'%.5f,',ET1(i,j));
        end
    end
end

fprintf(datos,'\t\t\t{');%ET2
for i=1:n+m
    if i~=1 fprintf(datos,'\t\t    ');end
    for j=1:2*n
        if i==n+m && j==2*n
            fprintf(datos,'%.5f},\n',ET2(i,j));
        elseif j==2*n && i==1
            fprintf(datos,'%.5f,\t\t/*ET2*/\n',ET2(i,j));
        elseif j==2*n
            fprintf(datos,'%.5f,\n',ET2(i,j));
        else
            fprintf(datos,'%.5f,',ET2(i,j));
        end
    end
end

fprintf(datos,'\t\t\t{');%Qfis
for i=1:((N+1)*n)
    if i~=1 fprintf(datos,'\t\t    ');end
    for j=1:((N+1)*n)
        if i==((N+1)*n) && j==((N+1)*n)
            fprintf(datos,'%.5f},\n',Qfis(i,j));
        elseif j==((N+1)*n)
            fprintf(datos,'%.5f,\n',Qfis(i,j));
        else
            fprintf(datos,'%.5f,',Qfis(i,j));
        end
    end
end

fprintf(datos,'\t\t\t{');%Lo
for i=1:n+p
    if i~=1 fprintf(datos,'\t\t    ');end
    for j=1:p
        if i==n+p && j==p
            fprintf(datos,'%.5f},\n',Lo(i,j));
```

```
        elseif j==p && i==1
            fprintf(datos,'%.5f,\t\t/*Lo*/\n',Lo(i,j));
        elseif j==p
            fprintf(datos,'%.5f,\n',Lo(i,j));
        else
            fprintf(datos,'%.5f,',Lo(i,j));
        end
    end
end

fprintf(datos,'\t\t\t{');%L
for i=1:n+m
    if i~=1 fprintf(datos,'\t\t    ');end
    for j=1:p
        if i==n+m && j==p
            fprintf(datos,'%.5f},\n',L(i,j));
        elseif j==p && i==1
            fprintf(datos,'%.5f,\t\t/*L*/\n',L(i,j));
        elseif j==p
            fprintf(datos,'%.5f,\n',L(i,j));
        else
            fprintf(datos,'%.5f,',L(i,j));
        end
    end
end
fprintf(datos,'\t\t\t{');%c
for i=1:2*(n+m)
    if i~=1 fprintf(datos,'\t\t    ');end
    for j=1:p
        if i==2*(n+m) && j==p
            fprintf(datos,'%.5f},\n',c(i,j));
        elseif j==p && i==1
            fprintf(datos,'%.5f,\t\t/*c*/\n',c(i,j));
        elseif j==p
            fprintf(datos,'%.5f,\n',c(i,j));
        else
            fprintf(datos,'%.5f,',c(i,j));
        end
    end
end
fprintf(datos,'\t\t\t{');%d
for i=1:2*(n+m)
    if i==1 fprintf(datos,'%.5f,\t\t/*d*/\n',d(i));
        elseif i==2*(n+m) fprintf(datos,'\t\t    %.5f},\n',d(i));
        else fprintf(datos,'\t\t    %.5f,\n',d(i));
    end
end
fprintf(datos,'\t\t\t{0.0,0.0,0.0,0.0},\t\t/*x_gorro*/\n');
fprintf(datos,'\t\t\t{0.0,0.0,0.0,0.0},\t\t/*xk*/\n');
fprintf(datos,'\t\t\t{0.0,0.0},\t\t/*dr_gorro*/\n');
fprintf(datos,'\t\t\t{0.0,0.0},\t\t/*dr*/\n');

fprintf(datos,'\t\t\t{');%Xf
for i=1:n
    if i==1 fprintf(datos,'%.5f,\t\t/*Xf*/\n',Xf(i));
        elseif i==n fprintf(datos,'\t\t    %.5f},\n',Xf(i));
        else fprintf(datos,'\t\t    %.5f,\n',Xf(i));
    end
end

fprintf(datos,'\t\t\t{');%Uf
```

```
for i=1:m
    if i==1 fprintf(datos,'%.5f,\t\t/*Uf*/\n',Uf(i));
        elseif i==m fprintf(datos,'\t\t    %.5f},\n',Uf(i));
        else fprintf(datos,'\t\t    %.5f,\n',Uf(i));
    end
end

fprintf(datos,'\t\t\t5,\t\t/*Tm (modelo No lineal)*/\n');
fprintf(datos,'\t\t\t0.01,\t\t/*h (modelo No lineal)*/\n');
fprintf(datos,'\t\t\t{0.0,0.0},\t\t/*r_rampa*/\n');
fprintf(datos,'\t\t\t%f,\t\t/*alpha*/\n',alpha);
fprintf(datos,'\t\t\t%f};\t\t/*ro*/\n',ro);


fclose(datos);
```

Apéndice B

# Códigos **C** para *Arduino*®

**Código B.1** Código C : Rutina general de llamada y lectura de referencias

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

// Declaración de la estructura generada por A.5

void setup() {
Serial.begin(9600);

}

void loop() {

    int i,i0,i1;
    static double r[p]={0.1337,0.1486};
    static double u[m]={0,0};
    static double yk[p];
    static int cambio=0;
    double cambio_ref;

    if (cambio!=1 && cambio!=2)
    {
        if (Serial.available() > 0) {

                cambio = Serial.parseInt();
                if(cambio==1 || cambio==2)
                {
                  Serial.print("Introduzca ahora el cambio en la referencia ");
                  Serial.println(cambio);
                }else{
                  Serial.println("Dato invalido. Introduzca 1 ó 2 para cambiar la
referencia deseada.");
                }

        }
    }
     if (cambio==1)
     {
        if (Serial.available() > 0) {

                cambio_ref = Serial.parseFloat();
                if(cambio_ref < 1.2 && cambio_ref > 0.2)
                {
                    r[0]=cambio_ref - Planta4T.Xf[0];
```

```
                    cambio=0;
                    Serial.println("Referencia modificada con exito.");
                }
                else
                {
                    Serial.println("Referencia   invalida.   La   altura   debe   estar
comprendida entre 0.2 y 1.2");
                }

        }
     }
     if (cambio==2)
     {
        if (Serial.available() > 0) {

                cambio_ref = Serial.parseFloat();
                if(cambio_ref < 1.2 && cambio_ref > 0.2)
                {
                    r[1]=cambio_ref - Planta4T.Xf[1];
                    cambio=0;
                    Serial.println("Referencia modificada con exito.");
                }
                else
                {
                    Serial.println("Referencia   invalida.   La   altura   debe   estar
comprendida entre 0.2 y 1.2");
                }

        }
     }


    PLANTA_NO_LINEAL(u,&Planta4T);
    for (i0 = 0; i0 < p; i0++) {
         yk[i0] = 0.0;
         for (i1 = 0; i1 < n; i1++) {
                yk[i0] += Planta4T.C[i0*n + i1] * Planta4T.xk[i1];
                                    }
                                }


    MPC(&Planta4T,r,u,yk);

}
```

**Código B.2** Código C : Rutina completa de control

```
void MPC(struct MPCData *Data, double r[p], double u[m], double yk[p])
{
  unsigned long time;
  unsigned long t2;
  static unsigned long dif=0;
  static int k=1;
  int i;

   time=millis();

    if((time+dif)>(k*(1000*(Data->Tm))))
```

```
    {
      k++;
      observador(u,yk,&Planta4T);
      SSTO(r,&Planta4T);
      Calculo_U(&Planta4T,u);

       for(i=0;i<m;i++){
          Serial.println(yk[i],5);              //Arduino-MATLAB
          Serial.println(Data->r_rampa[i],5);
          Serial.println(r[i],5);
          }
//       for(i=0;i<m;i++){                      //Arduino-MonitorSerie
//          Serial.print(yk[i],5);
//          Serial.print("\t");
//          Serial.print(u[i],5);
//          Serial.print("\t");
//          Serial.println(Data->dr[i],5);
//          if(i==m-1)Serial.println("");
//          }
      t2=millis();
      dif=t2-time;
    }
}
```

Es importante decir que, como expusimos al final de la **Sección 6.4**, la comunicación serial del *Arduino*® es de una única vía, i.e., si se desea comunicar con el MATLAB® no podrá utilizarse el Monitor Serie de *Arduino*® que es precisamente la herramienta que utlizamos para cambiar en línea la referencia. Es por ello que una de estas dos secciones debe estar comentada en el código según se desee.

**Código B.3** Código C : Simulador no lineal de la planta

```
void PLANTA_NO_LINEAL(double u[m], struct MPCData *Data)
{
  int i;
  int i0;
  int j;
  double b_Xk1;
  double Xk1[n],Xk[n],Uk[m];
  unsigned long time;
  static int k=0;

       time=millis();

  if(time>(k*(1000*(Data->h))))
  {
       k++;

       for (i = 0; i < n; i++) {
             Xk[i] = Data->xk[i]+Data->Xf[i];
       }
       for (i = 0; i < m; i++) {
             Uk[i] = u[i]+Data->Uf[i];
       }

       Xk1[0] = Xk[0] + Data->h * (((-0.00013104 * sqrt(19.62 * Xk[0]) + 9.2673E-5 *
sqrt(19.62 * Xk[2])) + 8.3333333333333331E-5 * Uk[0]) /0.03);
       Xk1[1] = Xk[1] + Data->h * (((-0.00015074 * sqrt(19.62 * Xk[1]) + 8.8164E-5 *
sqrt(19.62 * Xk[3])) + 0.00011111111111111112 * Uk[1]) / 0.03);
       Xk1[2]   =   Xk[2]   +   Data->h   *   ((-9.2673E-5   *   sqrt(19.62   *   Xk[2])
+0.00016666666666666666 * Uk[1]) / 0.03);
    Xk1[3]   =   Xk[3]   +   Data->h   *   ((-8.8164E-5   *   sqrt(19.62   *   Xk[3])
```

```
+0.00019444444444444443 * Uk[0]) / 0.03);
    for (j = 0; j < 4; j++) {
       b_Xk1 = Xk1[j];
       if (Xk1[j] < 0.0) {
         b_Xk1 = 0.0;
         }

       Xk[j] = b_Xk1;
       Xk1[j] = b_Xk1;
       }


    for (i = 0; i < n; i++) {
         Data->xk[i] = Xk[i]-Data->Xf[i];
         }

    }

}
```

**Código B.4** Código C : Observador

```
void observador(double u[m],double yk[p],struct MPCData *Data)
{
  double x_gorroant[n];
  int i;
  double b_A[n];
  int i0,i1;
  double c_A;
  double b_B[n];
  double b_yk[p];

  for (i = 0; i < n; i++) {
   x_gorroant[i] = Data->x_gorro[i];
  }

  for (i0 = 0; i0 < n; i0++) {
    b_A[i0] = 0.0;
    for (i1 = 0; i1 < n; i1++) {
      b_A[i0] += Data->A[i0*n + i1] * Data->x_gorro[i1];
    }

    b_B[i0] = 0.0;
    for (i1 = 0; i1 < m; i1++) {
      b_B[i0] += Data->B[i0*m + i1] * u[i1];
    }
  }

  for (i = 0; i < p; i++) {
    c_A = 0.0;
    for (i0 = 0; i0 < n; i0++) {
      c_A += Data->C[i*n + i0] * Data->x_gorro[i0];
    }

    b_yk[i] = (yk[i] - c_A) - Data->dr_gorro[i];
  }

  for (i = 0; i < n; i++) {
```

```
    c_A = 0.0;
    for (i0 = 0; i0 < p; i0++) {
      c_A += Data->Lo[i*p + i0] * b_yk[i0];
    }

    Data->x_gorro[i] = (b_A[i] + b_B[i]) + c_A;
  }

  for (i = 0; i < p; i++) {
    c_A = 0.0;
    for (i0 = 0; i0 < n; i0++) {
      c_A += Data->C[i*n + i0] * x_gorroant[i0];
    }

    b_yk[i] = (yk[i] - c_A) - Data->dr_gorro[i];
  }

  for (i = 0; i < p; i++) {
    c_A = 0.0;
    for (i0 = 0; i0 < p; i0++) {
      c_A += Data->Lo[(i*p + i0) + n*p] * b_yk[i0];
    }

    Data->dr_gorro[i] += c_A;
  }
}
```

**Código B.5** Código C : SSTO

```
void SSTO(double r[p],struct MPCData *Data)
{
    int i,k,cont;
    double aux,lambda;
    double r_gorro[p];


    double e[2*(m+n)];
    int indices[2*(n+m)];
    double REF[n+m];
    double b_L[n+m];

    for (k = 0; k < p; k++) {
        Data->dr[k]=((1-Data->alpha) * Data->dr[k] + Data->alpha * Data->dr_gorro[k]);
       r_gorro[k] = r[k] - Data->dr[k];
       Data->r_rampa[k] = (1 - Data->ro) * Data->r_rampa[k] + Data->ro * r_gorro[k];

     }


    for (k = 0; k < 2*(n+m); k++) {
    aux = 0.0;
    for (i = 0; i < p; i++) {
      aux += Data->c[k *p + i] *Data->r_rampa[i];
    }
    e[k] = aux - Data->d[k];
  }
  i = 0;
  cont = 0;
  lambda = 1.0E+6;
```

```
  for (k = 0; k < 2*(n+m); k++) {
    if (e[k] > 0.0) {
      indices[cont] = (int)k;
      i = 1;
      cont++;
    }
  }

  if (i == 0) {
    for (k = 0; k < n+m; k++) {
      REF[k] = 0.0;
      for (i = 0; i < p; i++) {
        REF[k] += Data->L[k*p + i] *Data->r_rampa[i];
      }
    }

    for (i = 0; i < n; i++) {
      Data->xr[i] = REF[i];
    }

    for (i = 0; i < m; i++) {
      Data->ur[i] = REF[i + n];
    }
  } else {
    for (i = 0; i <= (int)cont - 1; i++) {
      aux = 0.0;
      for (k = 0; k < p; k++) {
        aux += Data->c[(indices[i] *p + k) ] *Data->r_rampa[k];
      }

      aux = Data->d[indices[i]] / aux;
      if (aux < lambda) {
        lambda = aux;
      }
    }

    for (k = 0; k < n+m; k++) {
      b_L[k] = 0.0;
      for (i = 0; i < p; i++) {
        b_L[k] += Data->L[k *p + i] *Data->r_rampa[i];
      }

      REF[k] = lambda * b_L[k];
    }

    for (i = 0; i < n; i++) {
      Data->xr[i] = REF[i];
    }

    for (i = 0; i < m; i++) {
      Data->ur[i] = REF[i + n];
    }
  }
}
```

**Código B.6** Código C : Cálculo de la actuación

```
void Calculo_U(struct MPCData *Data, double u[m])
{
  double b[(N+1)*n];
  double f[n+m];
  int i,j,ii,jj;
  double Qxr[n];
  double Rur[m];
  int i0;
  int FIN,kk,k;

  double x[(N+1)*n];
  double x_ant[(N+1)*n];
  double y[(N+1)*n];
  double incr_y[(N+1)*n];
  double t,t_sig;
  double fsat[(n+m)*(N+1)];
  double aux1[n+m];
  double Ez_b[(N+1)*n];
  double norma;
  double aux=0.0;

for(i=0;i<n*(N+1);i++)
    {
        x[i]=0;
        y[i]=0;
        b[i]=0;
        x_ant[i]=0;
    }

    t=1.0;
    FIN=0;
    kk=1;


  for (i = 0; i < n; i++) {
    Qxr[i] = Data->Q[i] * Data->xr[i];
  }

  for (i0 = 0; i0 < m; i0++) {
    Rur[i0] = Data->R[i0] * Data->ur[i0];
  }


  for (i = 0; i < n+m; i++) {
    for (i0 = 0; i0 < n; i0++) {
      f[i0] = -Qxr[i0];
    }

    for (i0 = 0; i0 < m; i0++) {
      f[m + i0] = -Rur[i0];
    }


  }

  for (i = 0; i < n; i++) {
    b[i + N*n] = Data->xr[i];
  }
```

```
 for (i0 = 0; i0 < n; i0++) {
   b[i0] = Data->x_gorro[i0];
 }

while (!(FIN != 0)) {

   // Calculo de fsat
       // fsat=f-E'*y1
       for(i=0;i<n+m;i++)
       {
           //  ET1*y1
           aux1[i]=0;
           for(j=0;j<2*n;j++)
           {
               aux1[i]+=Data->ET1[i*2*n + j]*y[j];
           }
           fsat[i]=f[i]-aux1[i];
       }
       for(ii=n+m,jj=n;ii<N*(n+m);ii+=n+m,jj+=n)
       {
           for(i=0;i<n+m;i++)
           {
               //  ET2*y1
               aux1[i]=0;
               for(j=0;j<2*n;j++)
               {
                   aux1[i]+=Data->ET2[i*2*n + j]*y[jj+j];
               }
               fsat[ii+i]=f[i]-aux1[i];
           }
       }

 sat2(fsat,&Planta4T);

       for(i=0;i<n;i++)
       {
           // I*zmpc-b
           Ez_b[i]=Data->zmpc[i]-b[i];
           // [A B]*zmpc-xr(termino final b)
           aux1[i]=0;
           for(j=0;j<n+m;j++)
           {
               aux1[i]+=Data->AB_I[i*(2*n+m) + j]*Data->zmpc[(n+m)*(N-1)+j];
           }
           Ez_b[n*N+i]=aux1[i]-b[i+N*n];
       }
       for(ii=n,jj=0;ii<n*N;ii+=n,jj+=n+m)
       {
           for(i=0;i<n;i++)
           {
               // [A B -I]zmpc-0 (resto de terminos de b)
               aux1[i]=0;
               for(j=0;j<2*n+m;j++)
               {
                   aux1[i]+=Data->AB_I[i*(2*n+m) + j]*Data->zmpc[jj+j];
               }
               Ez_b[ii+i]=aux1[i];
           }
       }
```

```
        for(i=0;i<(N+1)*n;i++){      aux+=Ez_b[i]*Ez_b[i];   }
        norma=sqrt(aux);
        if(norma <= 1E-6){FIN=1;}
        else{

          for(i=0;i<(N+1)*n;i++){
            incr_y[i]=0.0;
            for(j=0;j<(N+1)*n;j++){
              incr_y[i]+=-Data->Qfis[i*(N+1)*n + j]*Ez_b[j];
            }
            x[i]=y[i]+incr_y[i];
          }

          t_sig=0.5*(1+sqrt(1+4*t*t));
          for(i=0;i<(N+1)*n;i++){ y[i]=x[i]+((t-1)/t_sig)*(x[i]-x_ant[i]);
                                  x_ant[i]=x[i];}
          kk++;
          t=t_sig;
        }

    if (kk > 100) {
      FIN = 1;

      //     exceso de bucle
    }
  }//while


  for (i = 0; i < m; i++) {
    u[i] = Data->zmpc[i + n];
  }
}
```

**Código B.7** Código C : Saturación del problema de optimización

```
void sat2(double f[(n+m)*(N+1)], struct MPCData *Data)
{
  int loop_ub;
  int i0;
  int hor;
  double z;
  double x;
  double u1;



for (hor=0;hor<N;hor++){
    for (loop_ub = 0; loop_ub <= (n+m) - 1; loop_ub++) {
       if(loop_ub<n){ z = -f[loop_ub+(n+m)*hor] / Data->Q[loop_ub]; }
       else{ z = -f[loop_ub+(n+m)*hor] / Data->R[loop_ub-n] ; }
       x = z;
       if (z > 0.0) {
             x = 1.0;
       } else if (z < 0.0) {
             x = -1.0;
       } else {
             if (z == 0.0) {
             x = 0.0;
       }
```

```
    }

    z = fabs(z);
    u1 = Data->lim[loop_ub];

    z = (z <= u1) ? z : u1;
    Data->zmpc[loop_ub+(n+m)*hor] = z * x;
  }
}
}
```

# REFERENCIAS


[1] Johansson, Karl H. *The quadruple-tankprocess.* IEEE Transactionson Control Systems Technology,vol 8, 2000.

[2] Atmel | SMART ARM-based MCU datasheet

[3] Martin, James. *Programming Real-time Computer Systems.* Englewood Cliffs, NJ: Prentice-Hall Inc., 1965.

[4] Beck, A, y M Teboulle. *A fast iterative shrinkage-thresholding algorithm for linear inverse problems.* SIAM J. Imaging Science., 2009.

[5] MATLAB Coder. Generate C and C++ code from MATLAB code

Rawlings, James B. y Mayne, David Q. *Model Predictive Control: Theory and Design*. Madison, Wisconsin. Nob Hill Publishing LLC., 2009.

Limón, Daniel. *Tracking Model Predictive Control.* Universidad de Sevilla.

https://www.wikipedia.org